In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

University of Alberta

Analysis of Software Engineering Data Using Computational Intelligence Techniques

By

Gabriel Jarillo Alvarado

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta

Fall 2003

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisisitons et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-87789-2
Our file Notre référence
ISBN: 0-612-87789-2

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

University of Alberta

Library Release Form

Name of Author: Gabriel Jarillo Alvarado

Title of Thesis: Analysis of Software Engineering Data Using Computational Intelligence Techniques

Degree: Master of Science

Year this Degree Granted: 2003

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholary or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis not any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Date: May 16, 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommended to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Analysis of Software Engineering Data Using Computational Intelligence Techniques submitted by Gabriel Jarillo Alvarado in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Witold Pedrycz (Co-supervisor)

Dr. Marek Reformat (Co-Supervisor)

Dr. Petr Musilek

Date: April 29, 2003

Dr. Jozef Szymanski

Abstract

This work aims at predicting the number of defects of Object Oriented (OO) software using Computational Intelligence techniques. There are 6 software metrics, also known as "CK metrics" to use as inputs for the prediction system, and the number of modifications made to the software projects as their output values. The CK metrics and number of Lines of Code of 5 software projects are available for this work, they are to be used to generate a system capable of determining the number of modifications made in the software projects based on their CK metrics.

The techniques to use in this work are: Fuzzy Clustering, Multivariable regression, Clustering and Local Regression, Neural Networks, Switching Regression Models and Fuzzy Clustering, and Genetic Algorithm - Based Clustering Method. At the end the different method are compared and discussed.

ACKNOWLEDGMENTS

The author wishes to express most sincere appreciation to Dr. Marek Reformat and Dr. Witold Pedrycz for their assistance, guidance and wise advices in the preparation of this manuscript. In addition, the author would like to thank to Dr. GianCarlo Succi whose familiarity with the needs and ideas of the project was helpful during the early Planning stage of this study. Thanks also to the members of the QuaSE Laboratory, University of Alberta for their valuable input.

TABLE OF CONTENTS

CHAPTER 1 . INTRODUCTION 1

CHAPTER 2 . THE SOFTWARE DATA 4

2.1 DESCRIPTION OF THE DATA 4
2.2 MORPHOLOGY OF THE DATA 6

CHAPTER 3 . PREPROCESSING "CLUSTERING OF THE DATA" 15

3.1 BACKGROUND ON FUZZY *C* MEANS 15
3.2 EXPERIMENTATION AND RESULTS 17
3.3 CONCLUSIONS 18

CHAPTER 4 . MULTIVARIABLE REGRESSION 19

4.1 BACKGROUND ON MULTIVARIABLE REGRESSION 19
4.1.1 LEAST SQUARES ESTIMATION 19
4.1.2 MATRIX APPROACH TO MULTIPLE LINEAR REGRESSION 21
4.2 EXPERIMENTATION AND RESULTS 22
4.3 CONCLUSIONS 26

CHAPTER 5 . CLUSTERING AND LOCAL REGRESSION 27

5.1 BACKGROUND ON CLUSTERING AND LOCAL REGRESSION 27
5.2 EXPERIMENTATION AND RESULTS 29
5.3 CONCLUSIONS 37

CHAPTER 6 . NEURAL NETWORKS 38

6.1 BACKGROUND ON BACKPROPAGATION NEURAL NETWORKS 39
6.1.1 NEURON MODEL WITH UNIPOLAR SIGMOID FUNCTION 40
6.1.2 BACKPROPAGATION LEARNING RULE WITH A SINGLE HIDDEN LAYER 41
6.2 APPROACH ONE (FEED FORWARD NEURAL NETWORK) 45
6.2.1 EXPERIMENTATION AND RESULTS 45
6.2.2 CONCLUSIONS 46
6.3 APPROACH TWO (LEAVE ONE OUT TESTING METHOD) 47
6.3.1 EXPERIMENTATION AND RESULTS 47
6.3.2 CONCLUSIONS 48

6.4 Approach three (Constant weight modifiers) **49**
6.4.1 Experimentation and results 49
6.4.2 Conclusions 50
6.5 Approach four (Constant weight modifiers and consistent elements) **52**
6.5.1 Experimentation and results 52
6.5.2 Conclusions 52
6.6 Approach five (Constant weight modifiers and most correlated attributes) **53**
6.6.1 Experimentation and results 53
6.6.2 Conclusions 55
6.7 Discussion **56**

CHAPTER 7 . SWITCHING REGRESSION MODELS AND FUZZY CLUSTERING **59**

7.1 Background on Regression Models and Fuzzy Clustering **59**
7.2 Experimentation and Results **60**
7.3 Conclusions **63**

CHAPTER 8 . GENETIC ALGORITHM-BASED CLUSTERING METHOD **65**

8.1 Background on Genetic Algorithm-Based Clustering Method **65**
8.1.1 Methodology to Find Consistent Data sets and Their Models 67
8.1.2 Methodology to Cluster Inconsistent Data sets. 68
8.2 Background on Genetic Algorithms **69**
8.3 Experimentation and Results **71**
8.3.1 Experimentation With Artificial Data 71
8.3.2 Results for the experimentation with artificial data 73
8.3.3 Experimentation With Data From the Industry 77
8.3.4 Results for the Experimentation With Data From the Industry 77
8.4 Conclusions **80**

DISCUSSION **82**

BIBLIOGRAPHY AND WWW LINKS **86**

APPENDIX A **89**

Description and Morphology of the data **89**
Cross correlation of the software data sets 90
Histograms of the software CK-metrics 93

APPENDIX B **98**

RESULTS OF FUZZY C MEANS CLUSTERING 98

APPENDIX C 103

RESULTS OF CLUSTERING AND LOCAL REGRESSION 103

APPENDIX D 117

SWITCHING REGRESSION MODELS AND FUZZY CLUSTERING 117

LIST OF TABLES

Table 1: Number of observations per project....6
Table 2: Mean and Standard Deviation of the Data Sets....7
Table 3: Correlation of the software metrics to their corresponding number of fixes....8
Table 4: Cross correlation for all metrics for project D....10
Table 5: Proportion of zero-inflation in the data sets....11
Table 6: Distribution of repeated attributes with different number of fixes....11
Table 7: Coefficients of the linear models....23
Table 8: Averaged squared differences....25
Table 9: Regression coefficients for Project A considering 6 clusters....31
Table 10: Error of the regression models in each cluster in each dimension....34
Table 11: Valid clusters to build multivariate models....35
Table 12 characteristics of the best NNs in approach one....45
Table 13: characteristics of the best NNs in approach two....47
Table 14 characteristics of the best NNs in approach three....50
Table 15 characteristics of the best NNs in approach four....52
Table 16 characteristics of the best NNs in approach five....55
Table 17: Best number of neurons for each data set....57
Table 18: Constant weight modifiers in each data set....58
Table 19: Regression models and errors for each cluster in project B....62
Table 20: Characteristics for X_{iM}....73
Table 21: Models and fitness function....75
Table 22: Characteristics of X_{iM}....77
Table 23: Averaged squared differences....78
Table 24: Coefficients of the linear models....79

LIST OF FIGURES

Figure 1: Data flow for the computational intelligence models 5
Figure 2: Standard deviation for all the internal software metrics in the data sets 8
Figure 3: Correlation between the internal metrics to the number of fixes for all projects 9
Figure 4: Cross correlation for all CK metrics for project D 10
Figure 5: Data NoM of project A 12
Figure 6: Data DIT of project A 12
Figure 7: Data NOC of project A 12
Figure 8: Data CBO of project A 12
Figure 9: Data RFC of project A 13
Figure 10: Data LCOM of project A 13
Figure 11: Histogram for NoM of project A 14
Figure 12: Histogram for DIT of project A 14
Figure 13: Histogram for NOC of project A 14
Figure 14: Histogram for CBO of project A 14
Figure 15: Histogram for RFC of project A 14
Figure 16: Histogram for LCOM of project A 14
Figure 17: Distribution of zero data for 2 clusters 17
Figure 18: Proportion of zero data for 2 clusters 17
Figure 19: Distribution of zero data for 4 clusters 17
Figure 20: Proportion of zero data for 4 clusters 17
Figure 21: Distribution of zero data for 6 clusters 18
Figure 22: Proportion of zero data for 6 clusters 18
Figure 23: Consistency for Project A 23
Figure 24: Consistency for Project B 23
Figure 25: Consistency for Project C 24
Figure 26: Consistency for Project D 24
Figure 27: Consistency for Project E 24
Figure 28: Squared error of the regression models 25
Figure 29: Averaged squared differences or MSE 26
Figure 30: Clustering and local regression 28
Figure 31: Clusters to analyze for project A 30
Figure 32: Clusters to analyze for project B 30
Figure 33: Clusters to analyze for project C 30
Figure 34: Clusters to analyze for project D 30
Figure 35: Clusters to analyze for project E 31
Figure 36: Model in cluster 1 for NoM 32
Figure 37: Model in cluster 1 for DIT 32
Figure 38: Model in cluster 1 for NOC 32
Figure 39: Model in cluster 1 for CBO 32
Figure 40: Model in cluster 1 for RFC 33
Figure 41: Model in cluster 1 for LCOM 33

Figure 42: Linear models for NoM of project A 33
Figure 43: Linear models for DIT of project A 33
Figure 44: Linear models for NOC of project A 34
Figure 45: Linear models for CBO of project A 34
Figure 46: Normalized fixes Vs Predicted *y* 36
Figure 47: Standardize residuals 36
Figure 48: Histogram and distribution of residuals 37
Figure 49: Neuron model with unipolar sigmoid function 40
Figure 50: Sigmoid activation function 41
Figure 51: Feedforward neural network diagram 42
Figure 52: R^2 of attributes to fixes in project A 53
Figure 53: R^2 of attributes to fixes in project B 53
Figure 54: R^2 of attributes to fixes in project C 54
Figure 55: R^2 of attributes to fixes in project D 54
Figure 56: R^2 of attributes to fixes in project E 54
Figure 57: Summary of the NN behaviour 57
Figure 58: Sensitivity of the switching regression models and fuzzy clustering 61
Figure 59: Regression models for project B, view from 6^{th} dimension 63
Figure 60: Delta set for *D* (consistent) 72
Figure 61: Delta set for *S* (inconsistent) 72
Figure 62: Performance of the clustering algorithm with artificial data 74
Figure 63: Accuracy for 2 clusters 76
Figure 64: Accuracy for 5 clusters 76
Figure 65: Accuracy for 10 clusters 76
Figure 66: Accuracy for *n* clusters 76
Figure 67: Errors of software projects according to number of clusters 78
Figure 68: Averaged squared differences or MSE 78
Figure 69: Consistency for project A 79
Figure 70: Consistency for project B 79
Figure 71: Consistency for project C 80
Figure 72: Consistency for project D 80
Figure 73: Consistency for project E 80
Figure 74: Comparative chart for NN, Multivariable Regression, and GA-Based Clustering Method 83

LIST OF EQUATIONS

Equation 1: Correlation equation....9
Equation 2: Covariance equation....9
Equation 3: General linear regression model....19
Equation 4: Least squares function....19
Equation 5: Condition 1 for least squares estimates....20
Equation 6: Condition 2 for least squares estimates....20
Equation 7: Simplified conditions for least squares estimates....20
Equation 8: Linear model in matrix notation....21
Equation 9: Lease squares normal equations in matrix form....22
Equation 10: Least squares estimate of β....22
Equation 11: Fitted regression model....22
Equation 12: Fitted regression model in matrix notation....22
Equation 13: Squared Error....24
Equation 14: Overall goodness of the clusters....28
Equation 15: Euclidean distance....29
Equation 16: Mean Squared Error....34
Equation 17: sigmoid function....41
Equation 18: Total sum-squared error (TSSE)....42
Equation 19: Total mean-squared error (TMSE)....43
Equation 20: Estimation of *w*....44
Equation 21: Estimation of **w** in matrix form....44
Equation 22: Estimation of *u* (weights in output layer)....44
Equation 23: Estimation of *w* (weights in hidden layer)....45
Equation 24: Estimation of *w* (weights in hidden layer) with constant weight modifier....49
Equation 25: Form of the regression model....59
Equation 26: Generalized form of the regression model....59
Equation 27: Error of the Regression Models and Fuzzy Clustering....60
Equation 28: General model....66
Equation 29: Proposed model example....67
Equation 30: Fitness Function....68
Equation 31: Proposed model....71
Equation 32: Artificial data set....72
Equation 33: Proposed model for software data....77

Chapter 1. Introduction

Software is playing an ever-increasing role in today's society and in the industry. Modern software organizations operate in a high dynamic market, under tight time and cost constraints. As an answer to these business and market needs, organizations have started to undertake software process improvement (SPI) initiatives aimed at increasing the maturity and quality of their software processes. Investment in process improvement has a significant business benefits such as refining the product quality, increased organizational flexibility and customer satisfaction [21].

The ultimate achievement of the software companies is to produce quality products that meet the necessities and requirements of the final user in an efficient and accurate way. When the software requirements and specifications become complex, the software engineers face the challenge of designing a product that is reliable, useful and simple to use, however that is not an easy task, very commonly the final software products contain bugs that sometimes become critical in their performance to the point of redesigning the software.

As a result of such problems, the companies are trying to use models to estimate the amount of defects or bugs that their products will have so they can invest the appropriate resources to correct them; at the end, accurate models would help them to produce better software with lesser defects and perhaps a better design.

As an example of the importance of modeling software engineering data, the accurate estimation of software development effort has major implications for the management of software development in the industry. Underestimates lead to time pressures that may compromise full functional development and thorough testing of the software product. On the other hand, overestimates can result in over allocation of development resources and personnel [15]. Many models for effort estimation have been developed during the past years; some of them use parametric methods with some degree of success, other kind of methods belonging to the computational intelligence family, such as Neural Networks (NN), have been also studied in this field showing more accurate estimations, and finally the Genetic programming (GP) techniques are being considered as promising tools for the prediction of effort estimation.

Organizations are also wondering how they can predict the quality of their software before it is used. Generally there are three approaches to do so [9]:

1. Predicting the number of defects in the system.

2. Estimating the reliability of the system in terms of time and failure.

3. Understanding the impact of the design and testing processes on defect counts and failure densities.

Knowing the quality of the software allows the organization to estimate the amount of resources to be invested on its maintenance. Software maintenance is a factor that consumes most of the resources in many software organizations therefore it's worth it to be able to characterize, assess and predict defects in the software at early stages of its development in order to reduce maintenance costs. Maintenance involves activities such as correcting errors, maintaining software, and adapting software to deal with new environment requirements [20].

From these examples it becomes evident the need of having systems that allow us to predict defects in the software at early stages of its development. However, program comprehension is a complex task. The software engineer must examine both, the structural aspect of the software code (e.g., programming language syntax) and the nature of the problem domain (e.g., comments, documentation, and variable names) to extract the information needed to fully understand any part of a software system. A number of tools and methods have been investigated to address both aspects. In general, structural information is easy to extract, but the real problem is on how to utilize that information properly [11].

The main intention of this work is to predict the number of defects of the software using Computational Intelligence techniques. There are 6 CK metrics {Number of Methods (NoM), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM) to use as inputs for the prediction system, while the independent variable or output of the system is number of defects, also known as "fixes". The CK metrics, number of Lines of Code, and number of modifications of 5 software projects from the industry are available for this work, they are to be used to generate a system capable of determining the number of modifications made in the software projects based on to the their CK metrics.

To illustrate the importance of modeling software engineering data we can consider the following examples, they show also the relevance of the computational intelligence techniques in the software engineering field.

There have been different approaches for estimating software development effort using computational intelligence techniques. Krishnamoorty Srinivasan and Douglas Fisher studied the behavior of the Neural Network (backpropagation) and a variant of CART, called CARTX [15]. Robert Hochman, Taghi Khoshgoftaar and John P. Hudepohl explored the performance of the discriminant analysis and Evolutionary Neural Networks (ENN) [25], the ENN are a combination of NNs and Genetic Algorithms (GA). Carolyn Mair, *et al.* experimented with machine learning methods such as Artificial Neural Networks (ANNs), Case-Based Reasoning (CBR), and Rule Induction (RI) to obtain software effort estimates. They showed that ANN methods have superior accuracy than the rest of the methods [4]. Matthew Evett, Pei-der Chien, Taghi M. Khoshgoftaar, and Edward B. Allen

demonstrated that a Genetic Programming (GP) - based system does an excellent job on software quality prediction, and would be a useful tool for managers of large software projects [19].

Lionel C. Briand, Victor R. Basili, and William M. Thomas described a pattern recognition approach for analyzing software engineering data, called Optimized Set Reduction (OSR), which overcome some limitations of the classical data analysis techniques [18]. Renu Kumar, Suresh Rai, Jerry L. Trahan conclude that the neural network techniques are very useful tools in developing predictive models for identifying high-risk modules from software complexity metrics [24]. W. HSU and M.F. Tenorio discuss and demonstrate the use of Neural Network techniques for constructing software engineering effort models using the backpropagation and Self Organizing Neural Networks algorithms. Taghi M. Khoshgoftaar, and Robert M. Szabo showed that applying principal-components analysis to the raw data yields a NN model whose predictive quality is statistically better than a NN model developed using the raw data alone.

Several research projects implementing computational intelligence techniques have been performed using software metrics to determine software quality, as an example there is the work done by Taghi Khoshgoftaar, *et al.* [19] where they implemented Genetic Algorithms (GA) based system for targeting software modules for reliability enhancements.

Statistical approaches have been also proposed for the prediction of faults in the software; Khoshgoftaar and Munson have applied principal component analysis, regression analysis, and discriminant analysis to predict the quality of the software modules [13].

A combination of Neural Networks and statistical methods have been used to estimate the quality of the software taking into account the use of software engineering data. Khoshgoftaar, *et al.* [14] combined the principal component analysis with a backpropagation neural network, they concluded that such combination does not always warrantee better results than the neural networks alone or any other statistical method. These early studies also showed that NN models using only a single hidden layer of neurons have better predictive quality than some statistical models when predicting the number of faults [14]; just to mention some implemented approaches on the field of software engineering data modeling.

Chapter 2. The Software Data

This chapter provides an insight of the software engineering data to use with the computational intelligence techniques. It shows statistical analysis and a general picture of the data sets.

2.1 Description of the Data

The data used in this work comes from a major Canadian company. The data represents the software metrics obtained from five releases of telecommunications software systems; this software was developed in C++ programming language. The five datasets contain, as internal metrics, the number of lines of code (LOC) and 6 Chidamber and Kemerer metrics, also known as (CK) metrics; and as external measure, the number of modifications made for each class. All the modifications of the classes, caused by defects in software operation, were recorded throughout the development process. The number of modifications is widely used as an estimator of the defect-proneness of a class in several scientific studies [10]. The set of CK metrics available for this study is compound by: number of methods (NoM), depth of inheritance tree (DIT), number of children (NOC), coupling between object (CBO), response for a class (RFC), and lack of cohesion in methods (LCOM). These CK metrics are explained later in this chapter.

Since the software metrics such as the CK metrics reveal the internal structural properties of the software, they are widely used as measures to estimate the quality and complexity of the software in the object oriented domain.

The CK object oriented design metrics and the source lines of code counts were collected from the source code using a WebMetrics tool, a software metrics collection system described in [26].

The following paragraphs provide a description of the internal and external metrics and their importance to characterize the software projects. Each project's data contains the described metrics per class and the number of modifications made to the same classes, in this way the data is compounded by a matrix of n rows and m columns. The rows present the classes used in the C++ projects while the columns present all the internal and external measures for such classes.

Figure 1 shows the inputs and outputs to be used in the modeling processes. The number of lines of code is not to be used as part of the input data set since it can only be obtained after the projects are finished, and the main goal of this work is to build models for prediction purposes.

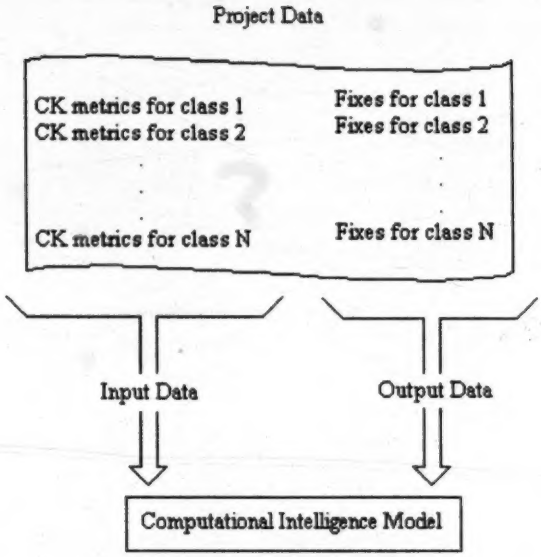


Figure 1: Data flow for the computational intelligence models

The metrics and measures used in these data sets are:

Internal metrics:

Lines of Code (LOC): this measure represents the number of lines of code written for a certain class.

For the CK metrics:

Number of Methods (NoM): This metric provides the number of methods in the class definition. In this project, this metric was used as a simplified version of the more general Weighted Method Count (WMC), as usually done [1].

Depth of Inheritance Tree (DIT): It's defined as the length of the longest path from the class to the root in the inheritance hierarchy [8].

Number of Children (NOC): It contains the number of classes that inherit from a particular class (i.e., the number of classes in the inheritance tree down from a class) [8].

Coupling Between Object Classes (CBO): A class is coupled with another if the methods of one class use the methods or attributes of the other, or vice versa. In this definition, uses can mean as a member type, parameter type, method local variable type or cast. CBO is the number of other classes with which a class is coupled. It includes inheritance-based coupling (i.e., coupling between classes related via inheritance) [8].

Response for a Class (RFC): The response set of a class consists of the set M of the methods of the class, and the set of methods invoked directly by the methods in M (i.e., the set of methods that can potentially be executed in response to a message received by that class). RFC is the number of methods in the response set of a class [8].

Lack of Cohesion in Methods (LCOM): It measures the number of pairs of methods in the class that have no attributes in common, minus the number of pairs of methods that

do. If the difference is negative, the metric value is set to zero [8]. It is desirable to have this value as low as possible; this is an inverse measure of a class' cohesion.

External measure:
Number of modifications for a class: This is a measure of the number of modifications or changes made to each class after the projects are finished, throughout this text this measure will be also referred as "Fixes".
The numbers of fixes in the data sets are the output of the models, while the internal software metrics are the input to the systems.
As mentioned before, the data come from five telecommunication software projects built in the object-oriented domain; the following table provides a description of the data. For confidentiality reasons the data sets will be referred to as Dataset A, B, C, D, and E.

Project	A	B	C	D	E
Number of Classes (observations in each data set)	93	120	101	44	38

Table 1: Number of observations per project

2.2 Morphology of the data

The input data to the models are the software CK metrics for each class since those are the metrics available from the design phase of the software. The number of lines of code is a metric that can only be obtained after the code has been written, for this reason the LOC count cannot be part of the input data to the system if it is to predict the number of modifications before the software has been finalized.

As shown in Table 1, the number of observations in each project in terms of classes is below or equal to 120 points, which means that building the models for them will be a challenging task for the computational intelligence techniques. It is always desirable to have as many observations as possible so that the models can cover the morphology of the data and produce more accurate predictions.

Table 2 provides a better insight of the data sets, it contains the measures of the mean and the standard deviation for each project. There the metric or "attribute" LOC is included even though it's not of relevance to this work, but it serves as a reference for us to compare it to the CK metrics and to get an idea of the length of each class. It is known that the LOC reflects, with some degree of accuracy, the complexity of the software, and therefore in an indirect way, the probability of faulty proneness; for such reason it is, in some cases, considered as the output for some models.

		LOC	NoM	DIT	NOC	CBO	RFC	LCOM	Fixes
Project A	**Mean**	94.645	9.785	0.903	0.269	11.677	24.591	59.441	5.280
	Standard Deviation	128.415	9.241	1.262	1.385	12.102	25.788	115.316	4.689
Project B	**Mean**	54.133	9.167	0.958	0.158	4.133	17.442	87.167	2.333
	Standard Deviation	139.308	13.702	1.114	0.619	7.967	35.099	233.148	5.237
Project C	**Mean**	247.416	32.604	0.970	0.158	23.168	67.455	1041.297	1.376
	Standard Deviation	301.479	40.926	0.959	1.192	23.966	71.548	3181.725	3.102
Project D	**Mean**	120.818	16.091	0.250	0.136	11.045	33.227	256.568	1.295
	Standard Deviation	255.285	19.036	0.433	0.625	18.031	51.782	693.857	2.889
Project E	**Mean**	468.184	41.605	0.263	0.053	17.816	69.053	1132.342	5.553
	Standard Deviation	603.752	42.839	0.547	0.320	22.365	74.195	1802.670	6.938

Table 2: Mean and Standard Deviation of the Data Sets

The projects are significantly different from each other in terms of design, as shown in the previous table, therefore the datasets cannot be combined together to increase the number of observations. Figure 2 depicts the standard deviation for the internal software metrics for all projects. There it is possible to see that the software metric or "attribute" LCOM is the most scattered one, This could present a problem for the computational intelligence techniques since the mapping of the data could be difficult to achieve. As for the rest of the attributes, there are similarities except for LOC, but it isn't of importance since this attribute is not part of the inputs to the prediction model.

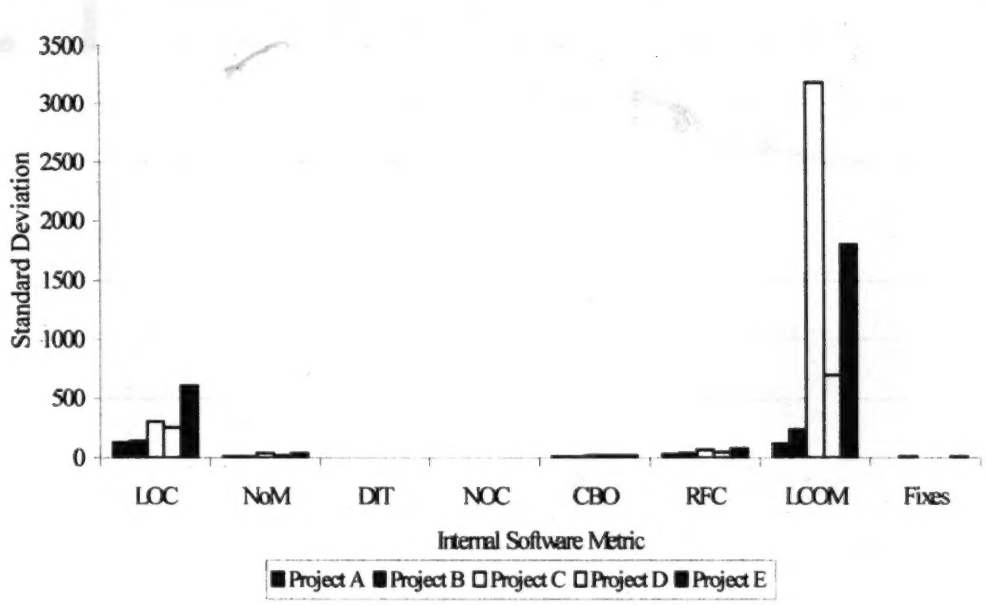


Figure 2: Standard deviation for all the internal software metrics in the data sets

It is important to consider the correlation between the software metrics (inputs) to the number of fixes (outputs) as well as the cross correlation between the input attributes themselves, this provides us with a better view of how different the inputs are with respect to the outputs. If the difference is too high it will be harder for the modeling techniques to converge to a solution since there are not similar data elements in the input space to the output domain. By looking at the cross correlation between the attributes we can appreciate which metric is the predominant one and by how much, this can highlight any noisy metric that could be affecting the accuracy of the models. Table 3 contains the correlation obtained for the internal software metrics with respect to the number of fixes.

Project	LOC	NoM	DIT	NOC	CBO	RFC	LCOM
A	0.1734	0.3121	-0.0009	-0.0894	0.1814	0.2984	0.4063
B	0.1975	0.11	0.3497	-0.0883	0.2476	0.2242	0.1237
C	0.5442	0.3031	0.0903	0.0000	0.4409	0.4149	0.2447
D	0.1686	0.2306	-0.0591	-0.0979	0.1943	0.2238	0.2291
E	0.7156	0.3027	0.0934	-0.1316	0.7361	0.6106	0.4226

Table 3: Correlation of the software metrics to their corresponding number of fixes

The correlations were calculated using:

$$Correlation = \frac{Cov(X, y)}{\sigma X \cdot \sigma y}$$

Equation 1: Correlation equation

Where,

$$Cov(X, y) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \mu_X)\cdot(y_i - \mu_y)$$

Equation 2: Covariance equation

n is the number of observations in the data set, i is the ith element of the data set, X denotes the input data set, and y is the output data set.

The correlations of Table 3 are plotted in Figure 3. There it is evident the lack of similarity between the attribute NOC to the number of fixes, this metric will challenge the computational intelligence techniques to converge to a solution. Also the attribute DIT has a low correlation in most of the projects with respect to the rest of the attributes. This suggest the option of discarding these two attributes from the input data sets to improve the prediction in the system, nevertheless they are considered in the experiments and, as a different approach, they can be discarded to explore the performance of the models without conflicting data. In general projects C and E are the most correlated to the number of fixes, this encourages us to expect the better results with these projects.

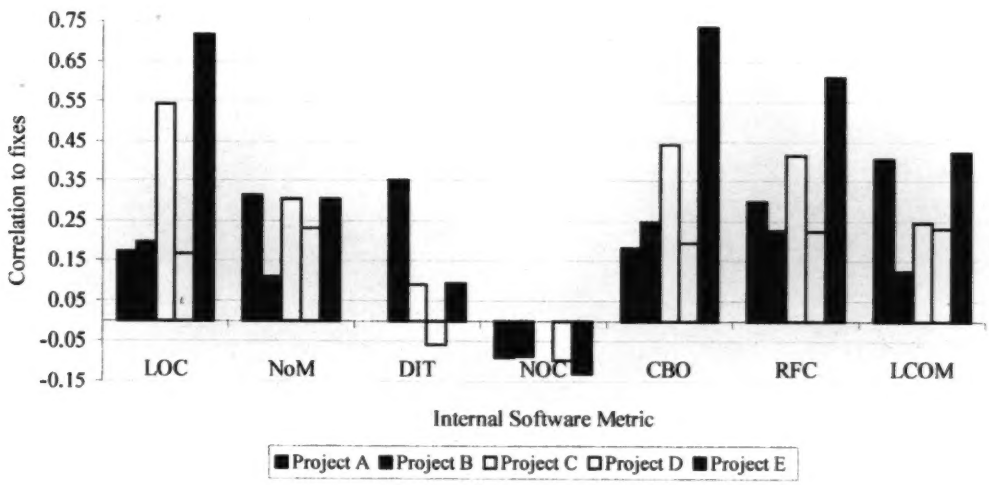


Figure 3: Correlation between the internal metrics to the number of fixes for all projects

As for the cross correlation of all software metrics, the attributes DIT and NOC are the less correlated as shown in Table 4 and Figure 4. The complete information for all correlations can be found in Appendix A. This situation is consistent to the correlations of the internal metrics with the number of fixes. The information suggests that DIT and NOC are non-consistent attributes and perhaps they can be hard to cover by the computational intelligence techniques such as the neural networks. Nevertheless this is just an insight of the data and it could be premature to discard them just now, experimentation is necessary to approve such decision.

	LOC	NoM	DIT	NOC	CBO	RFC	LCOM	Fixes
LOC	1							
NoM	0.70876	1						
DIT	0.15255	0.00551	1					
NOC	-0.09330	-0.11185	-0.12598	1				
CBO	0.95732	0.69834	0.07713	-0.12963	1			
RFC	0.97603	0.79892	0.10389	-0.11333	0.96476	1		
LCOM	0.8107	0.92814	0.10830	-0.07675	0.78799	0.86988	1	
Fixes	0.16856	0.23055	-0.05905	-0.09786	0.19434	0.22381	0.2290	1

Table 4: Cross correlation for all metrics for project D

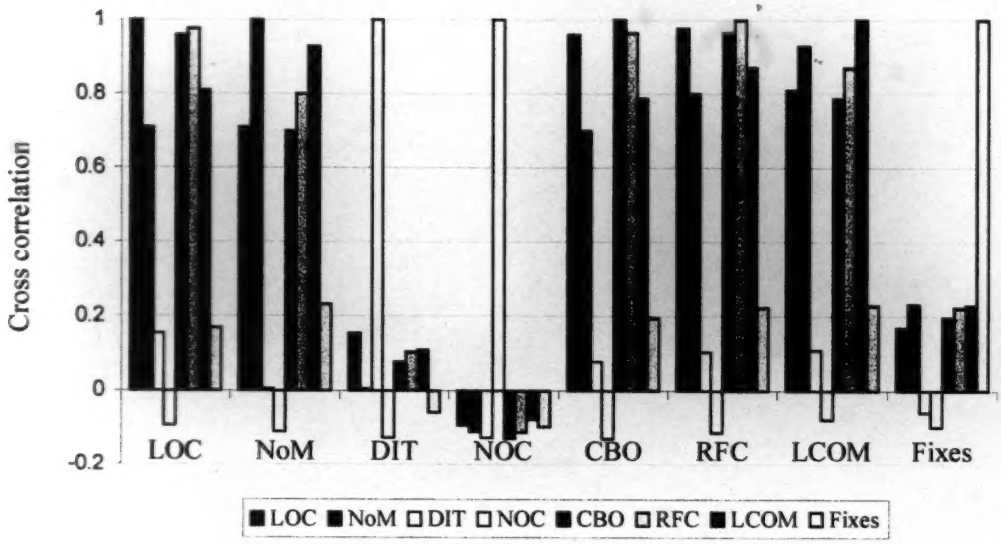


Figure 4: Cross correlation for all CK metrics for project D

Another important characteristic of the data to take into account is that it is zero inflated, most of the number of fixes are equal to zero in all projects, with the exception of project A. Projects C and D are good examples of this situation. This leads us to have even fewer

data points to train and test the models if these observations are discarded. From now on we will refer to such data interchangeably as "zero data", and to the sets that have values of fixes different than zero as "non-zero data".

Table 5 provides the proportion of the zeroes in the data sets of all five projects. In the case of data sets C and D the percentages of non-zero data elements are very low compare to the rest of the projects.

	Project A	Project B	Project C	Project D	Project E
% of zero data	0.00 %	55.83 %	64.35 %	70.45 %	28.94 %
% of non-zero data	100.00 %	44.17 %	35.64 %	29.54 %	71.05 %

Table 5: Proportion of zero-inflation in the data sets

In addition, the data is also inconsistent in the sense that for the same software metric's values the number of fixes is different. For the computational intelligence modeling process this is a problem. These differences between the outputs can reach up to 7 times bigger values. This non-consistency will make the models inaccurate to some degree; any model will fail in the prediction of the number of fixes with these conflicting data. One way to avoid this problem is taking away the inputs that have different associated outputs, such observations can be considered as noisy data. Since it is impossible to know which output is the correct one, it is unfeasible to choose to keep one of them in the data set.

Table 6 provides details on the number and percentage of the internal software metrics that are equal but have different associated outputs. The most critical case is project B, where from the original 120 observations it ends up having only 82 members, 31.67% of the original data are inconsistent inputs for the prediction models.

Project	Total observations	Repeated inputs with different outputs	Different Inputs with different outputs	% of repeated inputs	% of not repeated inputs
A	93	10	83	10.75%	89.25%
B	120	38	82	31.67%	68.33%
C	101	6	95	5.94%	94.06%
D	44	0	44	0.00%	100.00%
E	38	2	36	5.26%	94.74%

Table 6: Distribution of repeated attributes with different number of fixes

Since the input data consists of six attributes it is not possible to plot it in a single chart, nevertheless each dimension can be plotted against the number of fixes for that particular project to appreciate their morphology in a visual way.

Figure 5 to Figure 10show examples of the morphology of the data for project A, there it is possible to see the dramatic difference in the distribution of the data attributes in the space, for instance, dimensions 2 and 3 have most of their elements associated to very few values for the CK metrics (independent axis), therefore, it seems that there are not to many elements in these dimensions, nevertheless they are just overlapped to each other from these points of view. Such data may provide help to the prediction models.

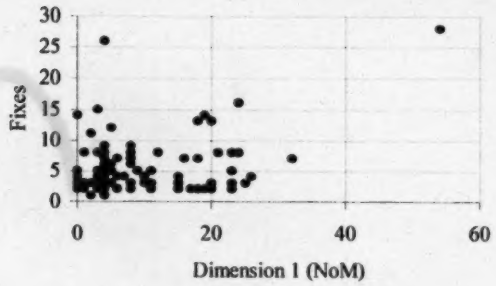


Figure 5: Data NoM of project A

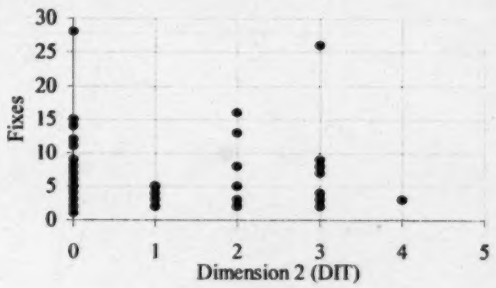


Figure 6: Data DIT of project A

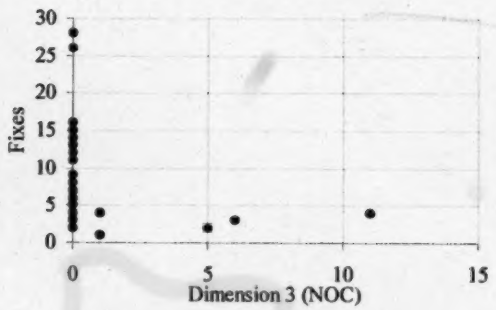


Figure 7: Data NOC of project A

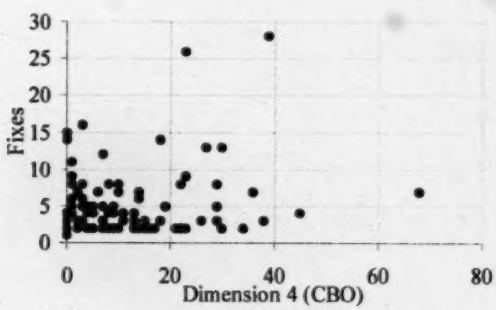


Figure 8: Data CBO of project A

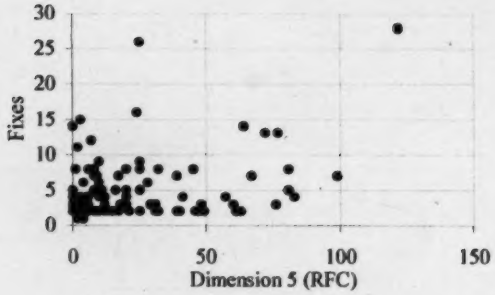


Figure 9: Data RFC of project A

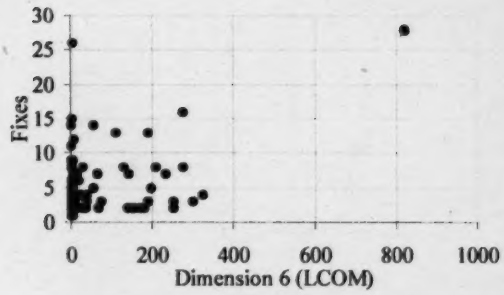


Figure 10: Data LCOM of project A

Also most of the data is concentrated close to the zero values, meaning that there were almost no fixes for small values in the CK metrics. This situation in fact makes sense because the lower the values of the CK metrics, the lower the complexity of the software, and therefore, the lower the probability of having errors or modifications for those particular classes. It is possible to see that in all dimensions the data are close to the origin of the plot, therefore most of the data is concentrated in that area considering all dimensions of the CK metrics.

It is important as well, to see the distribution of the data in each dimension to appreciate which software metrics provide useful information to the models. Figure 11 to Figure 16 show the histogram for each dimension (CK metric) of project A. If the data is spread out along to the independent axis, then the elements are different between themselves, but if the data is not widely distributed or just to close to each other, then the data points are not very different between themselves. It is desirable to have a very spread data along the independent axis as well as to have similar magnitudes for each data point in the histogram, so the modeling algorithms can differentiate between different clusters or groups of data.

It is evident that for project A, dimensions 2 and 3 are not providing much useful information to the modeling techniques, the data is not widely spread out along the independent axis and also the magnitude for the zero elements are very high compared to the rest of the elements. This demonstrates the zero-inflation phenomenon in the data set in the input domain as well. As for the 6^{th} dimension, most of the values are in the lower range; nevertheless the data is widely distributed along the independent axis according to its values. The histogram for all the datasets can be found in Appendix A.

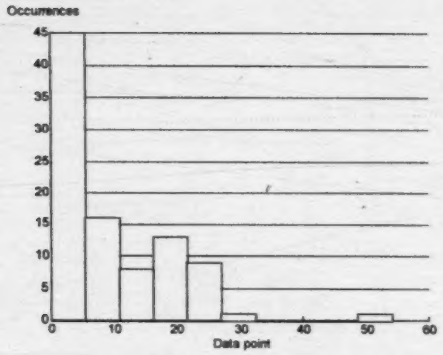


Figure 11: Histogram for NoM of project A

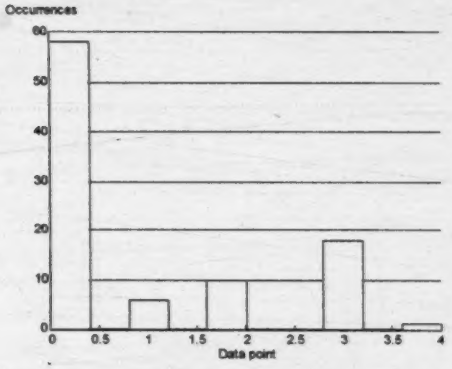


Figure 12: Histogram for DIT of project A

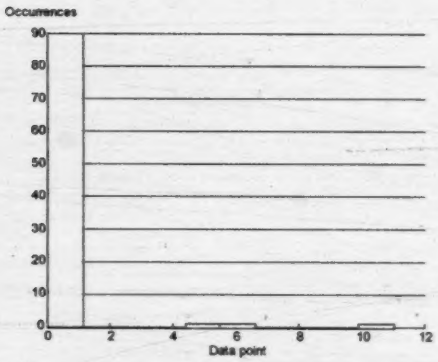


Figure 13: Histogram for NOC of project A

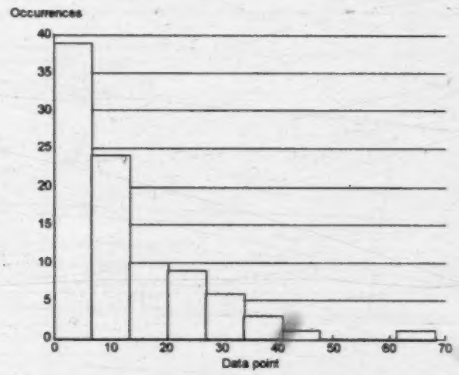


Figure 14: Histogram for CBO of project A

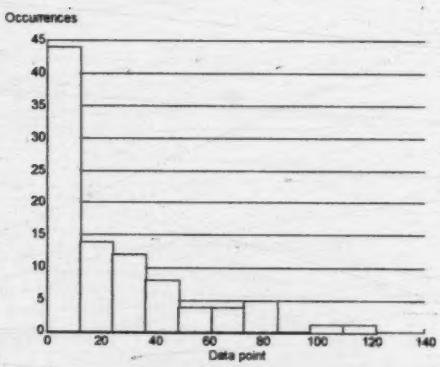


Figure 15: Histogram for RFC of project A

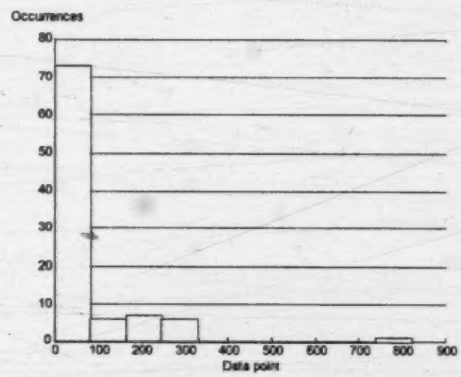


Figure 16: Histogram for LCOM of project A

Chapter 3. Preprocessing "Clustering of the data"

As it is described in Chapter 2, the software engineering data used for this work contains many zero values in the output set (number of fixes). This suggests, as a first attempt, the implementation of a clustering technique to, ideally, split the input space into 2 main sections, one containing all or most of the input observations (CK metrics) associated to the zero values in the output space (Fixes), and the other mainly having the input observations related to the non-zero values in the output. Having these two main groups can simplify the modeling of the data. The data elements belonging to the group associated to the zero values would not have to be modeled since the output would be always zero, therefore the models can be simplified and the computational effort can be minimized for the second group of data. According to this description, the clustering technique is used as a pre-processing part for the data, it has the main intention of simplifying the models later on, but this phase is not a requirement to build the models or to use any computational intelligence technique afterwards.

The Fuzzy C-means (FCM) clustering algorithm was used to group the data; its general description is presented in the next paragraphs *"Background on Fuzzy c Means"*. This is a very widely used clustering method in computational intelligence, data mining and knowledge discovery. The reader is encouraged to refer to [87] to find out details and related work for this algorithm. The Euclidean distance was used in this work to estimate the closeness of the observations in the data sets and to estimate the corresponding prototypes.

3.1 Background on Fuzzy *C* Means

The main goal of a clustering algorithm is to find groups or structures with similar elements in a data set; the clustering algorithm receives input data elements and returns clusters or classes of data according to their similarity. The clustering algorithm partitions the space according to certain prototypes and their distances to the rest of the elements in the data set.

The K-means clustering algorithm, also known as fuzzy *c*-means clustering, is based on the minimization of a performance index defined as the sum of all vectors in a cluster domain to the cluster center [87]. The general algorithm is compound of the following steps [87]:

Given a training data set containing M examples and *c* classes, user defined number of intervals n_{Fi} for feature F_i

1. For class c_j do, $j = 1, 2, \ldots, c$

2. Choose $K = n_{Fi}$ as the initial number of cluster centers.

3. Distribute the values of the feature among the K cluster centers "prototypes", based on the minimal distance criterion. As the result, feature values will cluster around the updated K cluster centers.

4. Compute K new cluster centers such that for each cluster the sum of the squared distance from all points in the same cluster to the new cluster center is minimized.

5. Check if the updated K cluster centers are the same as the previous ones, if yes go to step 1, otherwise go to step 3.

As a result, the final boundaries for the feature will be the minimum value the feature takes on, mid points between any two nearby cluster prototypes found for all clusters, and the maximum value the feature takes on.

It is necessary to know the membership of each element to each of the cluster in order to know to which group the element it belongs, the partition matrix u specifies these values for each element in the data set. In this particular case it is assumed that the sum of the membership of a specific feature is equal to 1, therefore

$$u = \{u_{ik} \in [0,1] \| \sum_{i=1}^{c} u_{ik} = 1\},$$

Where c denotes the clusters to find in the data set, and k is the kth element in the data set.

The partition matrix is obtained according to the following equation,

$$u_{ik} = \frac{1}{\sum_{i=1}^{c} \left(\frac{\|x_k - v_i\|}{\|x_k - v_j\|} \right)^2}; \ \forall k \in [1,2,\cdots,n] \text{ and } \forall i \in [1,2,\cdots,c],$$

Where $\|\cdot\|$ denotes distance, v_i is the ith cluster center, x_k is the kth element in the data set, n is the number of elements in the data set, and c is the number of clusters to be found.

The function to optimize the prototypes is,

$$v_i = \frac{\sum_{k=1}^{n} u_{ik} x_k}{\sum_{k=1}^{n} u_{ik}}$$

The fuzzy clustering aims to achieve a classification that is closer to the real world, because the object itself is usually of ambiguous, or fuzzy nature [22]. Under this framework the following tests were carried out.

3.2 Experimentation and Results

Figure 17 to Figure 22 show examples of some of the results obtained from the clustering algorithm for project B, it provides information about the quantity of zero data and non-zero data clustered in each group. The experiments were done considering 2, 4, 5, 6, and 7 clusters; they give an idea of how the clustering method performs in separating the zero from the non-zero data. The left bars depict the distribution of the data among the clusters while the right bars show the distribution of the data belonging to each cluster. The complete results for all projects can be found in Appendix B. Since data set A does not contain zeros, the clustering analysis is not necessary to split this set.

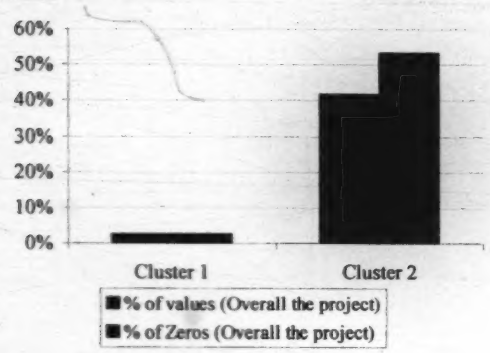


Figure 17: Distribution of zero data for 2 clusters

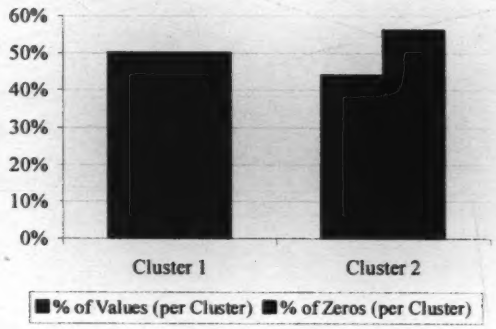


Figure 18: Proportion of zero data for 2 clusters

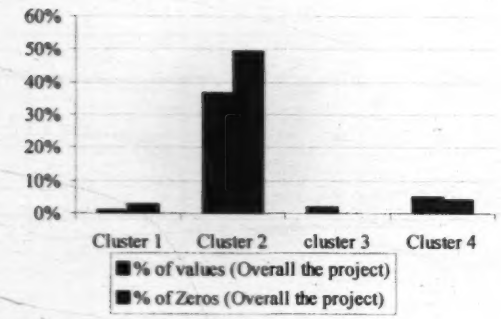


Figure 19: Distribution of zero data for 4 clusters

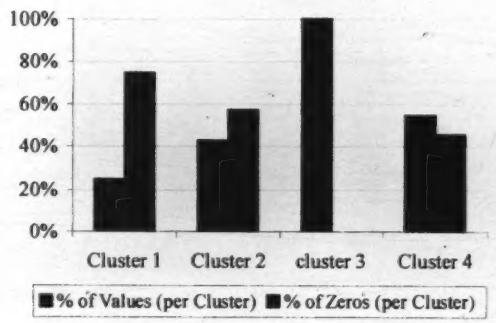


Figure 20: Proportion of zero data for 4 clusters

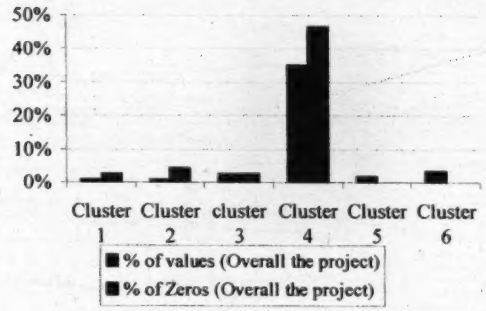


Figure 21: Distribution of zero data for 6 clusters

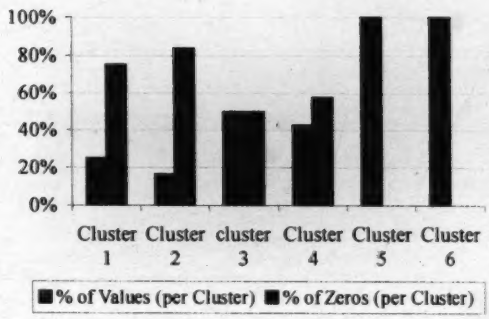


Figure 22: Proportion of zero data for 6 clusters

3.3 Conclusions

In the plots depicted from Figure 17 to Figure 22 it becomes evident that the clustering algorithm is not splitting the data into the two desired main categories (zero and non-zero data), it is possible to see from the charts in the left column that the data is mixed except in few cases where the clusters contain a small number of elements. The right column has the percentage of the data that are zero and non-zero elements in each cluster; the data is mixed up in similar proportions for most of the groups.

The results of the clustering algorithm indicate that the data is overlapped between the zero and non-zero elements, the nature and homogeneity of the sets prevents us from splitting the data into the two desired categories, therefore different approaches are needed to build the models. The main intention of the clustering algorithm is to simplify the construction of the models, having one group as zero data and another one with non-zero data, nevertheless it's not an impediment to create the models for the data sets.

Chapter 4. Multivariable Regression

Many application of regression models analysis involve situations where there are more than one variable to feed the models, like in this case. A regression model that contains more than one regression variable is called a "multiple regression model" or "multivariable regression".

4.1 Background on Multivariable Regression

In general, the dependant variable or response y may be related to M independent or regression variables. The linear multivariable model is of the form:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_M x_{iM} + \varepsilon = \beta_0 + \sum_{m=1}^{M} \beta_m x_{im} + \varepsilon; 1 \le i \le n$$

Equation 3: General linear regression model

This model is called a multiple linear regression model with M regressor variables and n observations. The parameters β_m, where $m = 0, 1, \ldots, M$, are called the regression coefficients. Such model describes a hyperplane in the m-dimensional space of the regression variables x_m. The parameter β_m represents the expected change in response y per unit change in x_m when all remaining regressors x_k ($k \neq m$) are held constant. Multiple linear regression models are often used as approximating function [7].

A multivariable regression analysis provides predictions based on the combined predictive effect of predictors. The method to obtain the regression coefficients for the software data is the "Least Squares Estimation".

4.1.1 Least Squares Estimation

The method of the least squares is used to estimate the regression coefficients in the multiple regression model of the form of Equation 3. Suppose that there are more observations than number of variables in the data set, this is, $n > M$, and let x_{im} represent the *ith* observation in the data set. The observations are $\{x_{i1}, x_{i2}, \ldots, x_{nM}, y_i\}$, where $n > M$.

The Least squares function is:

$$L = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{m=1}^{M} \beta_m x_{im} \right)^2$$

Equation 4: Least squares function

The function L is to be minimized with respect to $\beta_0, \beta_1, \ldots, \beta_M$. The least squares estimates of $\beta_0, \beta_1, \ldots, \beta_M$ must satisfy:

$$\left.\frac{\partial L}{\partial \beta_0}\right|_{\hat{\beta}_o, \hat{\beta}_1, \ldots, \hat{\beta}_M} = -2\sum_{i=1}^{n}\left(y_i - \hat{\beta}_o - \sum_{m=1}^{M}\hat{\beta}_m x_{im}\right) = 0$$

Equation 5: Condition 1 for least squares estimates

and

$$\left.\frac{\partial L}{\partial \beta_m}\right|_{\hat{\beta}_o, \hat{\beta}_1, \ldots, \hat{\beta}_M} = -2\sum_{i=1}^{n}\left(y_i - \hat{\beta}_o - \sum_{m=1}^{M}\hat{\beta}_m x_{im}\right) = 0; 1 \le m \le M$$

Equation 6: Condition 2 for least squares estimates

Simplifying the previous two conditions it is possible to obtain the least square normal equations:

$$\begin{aligned}
&n\hat{\beta}_0 + \hat{\beta}_1\sum_{i=1}^{n}x_{i1} + \hat{\beta}_2\sum_{i=1}^{n}x_{i1} + \cdots + \hat{\beta}_M\sum_{i=1}^{n}x_{iM} = \sum_{i=1}^{n}y_i \\
&\hat{\beta}_0\sum_{i=1}^{n}x_{i1} + \hat{\beta}_1\sum_{i=1}^{n}x_{i1}^2 + \hat{\beta}_2\sum_{i=1}^{n}x_{i1}x_{i2} + \cdots + \hat{\beta}_M\sum_{i=1}^{n}x_{i1}x_{iM} = \sum_{i=1}^{n}x_{i1}y_i \\
&\qquad\vdots \qquad\qquad\qquad\vdots \qquad\qquad\qquad\vdots \qquad\qquad\qquad\vdots \qquad\qquad\vdots \\
&\hat{\beta}_0\sum_{i=1}^{n}x_{iM} + \hat{\beta}_1\sum_{i=1}^{n}x_{iM}x_{i1} + \hat{\beta}_2\sum_{i=1}^{n}x_{iM}x_{i2} + \cdots + \hat{\beta}_M\sum_{i=1}^{n}x_{iM}^2 + \sum_{i=1}^{n}x_{iM}y_i
\end{aligned}$$

Equation 7: Simplified conditions for least squares estimates

There are $p = M + 1$ normal equations, one for each of the unknown regression coefficients. The solution of the normal equations are the least square estimators of the regression coefficients $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_M$. The normal equations can be solved by any method appropriate for solving a system of linear equations [7]. For this project, the matrix approach to multiple linear regression is used in order to solve the equations and to find the regression coefficients of the models.

4.1.2 Matrix approach to multiple linear regression

In fitting a multiple regression model, it is convenient to express the mathematical operations using matrix notation. The model is a system of n equations that can be expressed in matrix notation as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

Equation 8: Linear model in matrix notation

Where,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \cdots & x_{1M} \\ 1 & x_{21} & x_{22} \cdots & x_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \cdots & x_{nM} \end{bmatrix}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_M \end{bmatrix} \qquad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Normally, $\mathbf{y}$ is an (n x 1) vector of observations, $\mathbf{X}$ is an (n x ($M+1$)) matrix of the levels of the independent variables, $\boldsymbol{\beta}$ is a (($M+1$) x 1) vector of regression coefficients, and $\boldsymbol{\varepsilon}$ is an (n x 1) vector of random errors [7]. At the end the intension is to find the vector of least squares estimators $\hat{\boldsymbol{\beta}}$ that minimizes:

$$L = \sum_{i=1}^{n} \varepsilon_i^2 = \boldsymbol{\varepsilon}'\boldsymbol{\varepsilon} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

The least squares estimator $\hat{\boldsymbol{\beta}}$ is the solution for $\boldsymbol{\beta}$ in the equations:

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = 0$$

The resulting equation to be solved is:

$$\mathbf{X'X}\hat{\boldsymbol{\beta}} = \mathbf{X'y}$$

Equation 9: Lease squares normal equations in matrix form

Equation 9 are the least squares normal equation in matrix form, they are identical to the scalar form of the normal equations presented in Equation 7. Solving the normal equations requires multiplying both sides of Equation 9 by the inverse of **X'X**. Then the least squares estimate of **β** is:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X'X})^{-1}\mathbf{X'y}$$

Equation 10: Least squares estimate of β

There are $p = k + 1$ normal equations in $p = k + 1$ unknowns, which are the values of $\hat{\beta}_0, \hat{\beta}_1, \cdots, \hat{\beta}_M$.

The fitted regression model is:

$$\hat{y} = \hat{\beta}_0 + \sum_{m=1}^{M} \hat{\beta}_m x_{im}; 1 \le i \le n$$

Equation 11: Fitted regression model

Which in matrix notation is:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

Equation 12: Fitted regression model in matrix notation

In this way it is possible to estimate the values of the regression coefficients β for the multivariable linear models in the software data sets.

4.2 Experimentation and Results

The models proposed for the software data sets are of the form *fixes* = $\beta_0 + \beta_1 \mathrm{NoM}_i + \beta_2 \mathrm{DIT}_i + \beta_3 \mathrm{NOC}_i + \beta_4 \mathrm{CBO}_i + \beta_5 \mathrm{RFC}_i + \beta_6 \mathrm{LCOM}_i$. Lets also note that the output data are the number of fixes, however they will be also referred to simply as "*y*". Table 7 presents the coefficients of the linear models for each software project. Lets remember that the data is normalized in the rage [0, 1] in the input and output domains independently, therefore this coefficients provide a solution in such range.

Coefficients	*Project*				
	A	**B**	**C**	**D**	**E**
β_0	0.2040	0.0530	0.0170	0.0740	0.0582
β_1	-8.8526	-12.5830	-68.8523	-0.6506	-5.6876
β_2	-4.5026	123.0137	-283.7324	-170.2666	-191.4198
β_3	-6.3832	-65.0629	-73.9024	-106.7034	-361.3260
β_4	-4.2016	-16.1409	21.1785	-12.1947	38.3097
β_5	3.2598	8.8292	38.8256	6.0468	3.9015
β_6	0.8175	0.0250	0.3446	0.1101	0.1473

Table 7: Coefficients of the linear models

To better appreciate the accuracy of the models it is convenient to plot the results of the models vs. the actual number of modification provided in the data sets; if the models are completely accurate then the plots should depict a 45 degree line in the range [0, 1] since the data is normalized. Figure 23 to Figure 27 present the plots of the accuracy for all software projects.

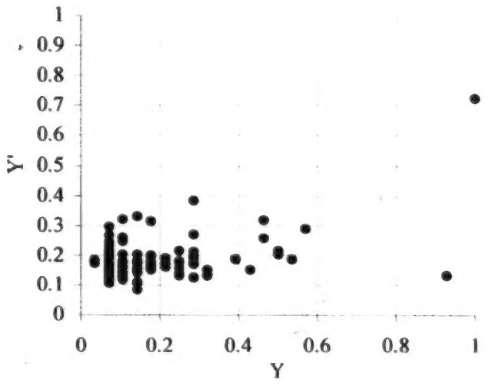


Figure 23: Consistency for Project A

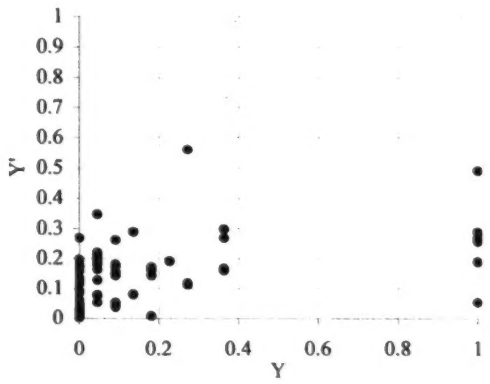


Figure 24: Consistency for Project B

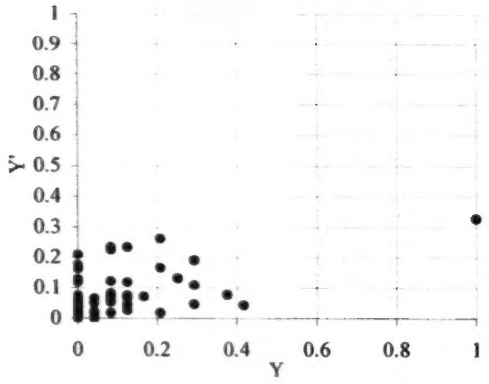


Figure 25: Consistency for Project C

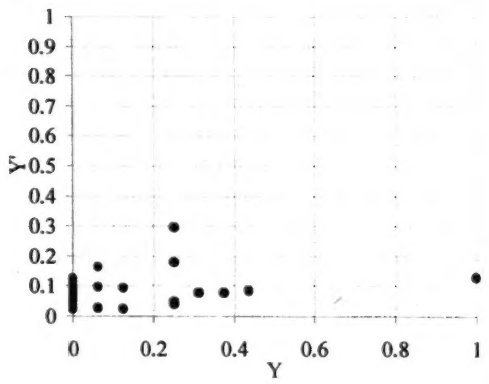


Figure 26: Consistency for Project D

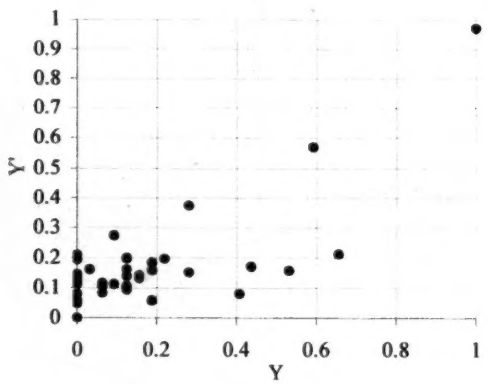


Figure 27: Consistency for Project E

The plots from Figure 23 to Figure 27 present some inaccuracy in the linear models, most of the elements are scattered near the origin of the plots, unfortunately the regression models are not good enough to describe the software data correctly, however some tendency of a 45 degree line can be appreciated in the consistency for project E.

Lets now estimate the squared error between the models' output and the actual number of modifications to appreciate the performance of the prediction models. Equation 13 is used to compute the error for the prediction models.

$$Squared_Error = \sum_{i=1}^{n} (y_i - y'_i)^2$$

Equation 13: Squared Error

Where, y is the actual number of fixes and y' denotes the result of the prediction models.

Figure 28 presents the errors of the regression models.

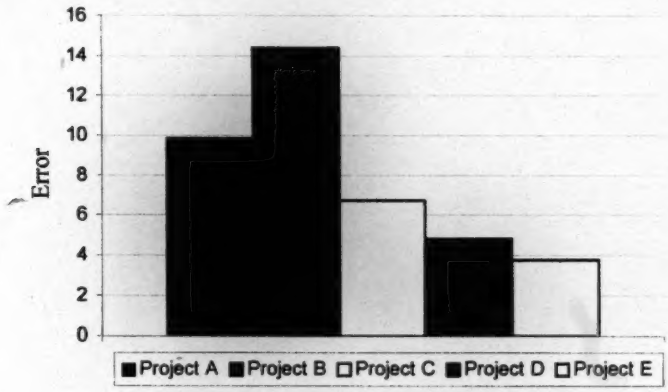


Figure 28: Squared error of the regression models

Notice that the Error is squared difference of the number of fixes using normalized values. The number of observations in the data sets play a role in the magnitudes in the previous plot, if there are considerably more elements in one project than there are in the others then is most likely that the Error increases, therefore an average of the squared differences for each project may provide a better understanding of the accuracy of the models, this is, obtaining the Mean Squared Error (MSE). Table 8 and Figure 29 present the average of the squared differences between the model's output and the actual number of fixes.

Project	A	B	C	D	E
MSE	0.1063	0.1197	0.0667	0.1100	0.0996

Table 8: Averaged squared differences

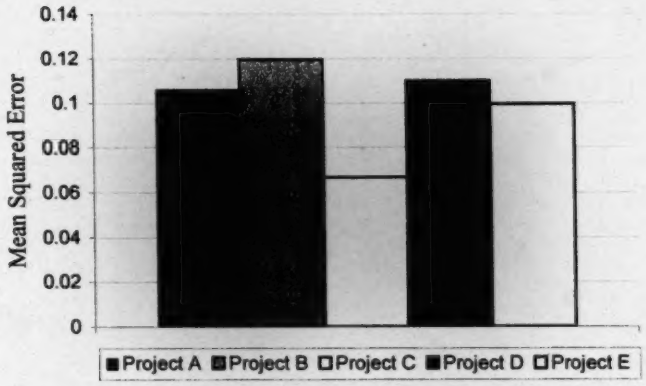


Figure 29: Averaged squared differences or MSE

The linear model for project C is the most accurate of all, even tough its plot of y vs. y' in Figure 25 does not depict an obvious 45 degree line, however the dots are scattered close to such imaginary line.

4.3 Conclusions

The multivariable regression technique is very useful to model data sets, in fact it predicts up to some degree the number of defects on the software data, this can be appreciated in the accuracy plots of Figure 23 to Figure 27.There are many inconsistent elements and several repeated observations with different associated outputs that prevent the linear models from describing the data sets. The results depicted in Figure 29 show that the models have similar accuracy for projects A, B, D, and E, while the most accurate model belongs to project C.

In Table 7, it is possible to appreciate the importance of the attributes DIT and NOC for projects B, C, D, and E; most of the coefficients of these attributes in the regression models are much higher "in magnitude" than the coefficients of the rest of the attributes, this is interesting because it happens that the correlation of these two attributes to the number of fixes is very poor, nevertheless the correlation of DIT and NOC to the number of fixes is negative for most of the models' projects. This makes sense because the regression models are actually minimizing the impact of these two attributes since they are poorly related to the number of fixes. The coefficient β_2, corresponding to the attribute DIT, in project B is the highest of all, confirming once again the intention of the regression models to adequate, in the best possible way, the impact of the attributes to the number of modifications in the software.

Chapter 5. Clustering and Local Regression

The main goal of this method is to find simple mathematical models capable and accurate enough to predict the number of defects out of the CK metrics. We have seen that the data is inconsistent and zero inflated, therefore it becomes evident that a simple model will not be able to cover the space with good precision. This situation leads to the idea of having several models to describe the complete data set. The task of choosing which data is to be used for a specific mathematical model is delivered to the clustering algorithm. If the data is broken down into several groups, each of them big enough to build a linear equation out of it, it is possible to have a multi-model representation of the sets. To be able to build a linear model only 2 data points are needed so, since the data sets are small, it could be inconvenient to build higher order models. Having a multiple model solution for the software data sets allows the possibility of providing different solutions (outputs) for the same input observations (CK-metrics).

The Fuzzy c- means clustering algorithm is used in this approach, it is effective in finding groups of data according to the distances between all the elements in the data sets, it also provides us with information about the probabilities for each element to belong to each cluster that is found. By combining the clustering algorithm and the linear regression method, it is possible to obtain a powerful method to describe the data in a comprehensible mathematical form, which is not the case for some other modeling methods such as neural networks.

5.1 Background on Clustering and Local Regression

Figure 30 depicts the general diagram of the clustering and local regression method implemented to this data. The input CK-metrics feed the FCM clustering algorithm, which finds the specified number of clusters and their corresponding prototypes, for now, the clusters' prototypes are not of importance to find the regression models. The clusters' data is then given to the linear regression algorithm to find the mathematical models for the particular group of elements, here the models are found using each dimension at a time from the CK-metrics set and the number of fixes. At the end, this complete algorithm offers several linear models to predict the number of fixes.

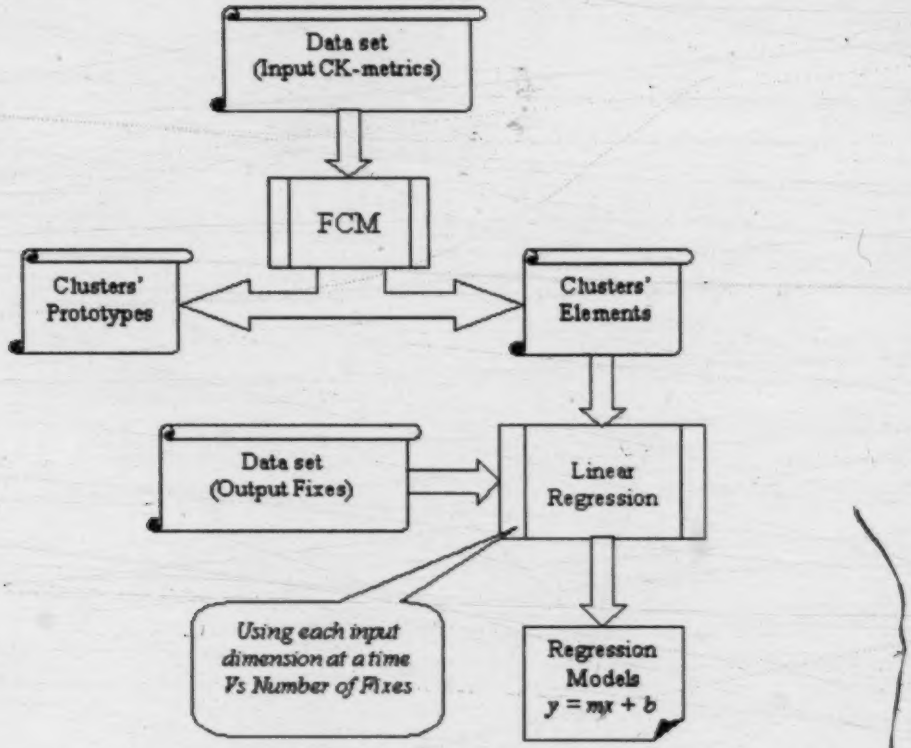


Figure 30: Clustering and local regression

The FCM algorithm requires a fixed number of clusters to find before it's execution, where the maximum number of groups cannot be grater the number of elements *n* belonging to the input dataset. In this project the clustering algorithm classifies the data into 2 to 30 groups, later the results are analyzed to find the best number of clusters for each particular data set. To measure the overall goodness of the clusters, the sum of the Euclidean distances for all clusters is computed according to Equation 14. Since the goodness is calculated only by the distances from the data elements to their prototypes, the smaller the overall summation of distances, the tighter the overall clusters are. It is to expect to find some low values for high number of clusters; nevertheless it is possible to find some local low values of the goodness of the clusters, and those are the candidates to be the best number of clusters to obtain for the data sets.

$$goodness = \sum_{i=1}^{c}\sum_{j=1}^{n}\left\|x_j - v_i\right\|^2$$

Equation 14: Overall goodness of the clusters

Where,

c = number of clusters,

n = number of data points in the cluster i,

$$\|x_j - v_i\| = \sqrt{\sum_{k=1}^{D} (x_{j,k} - v_{ik})^2}$$

Equation 15: Euclidean distance

D = Dimensionality of the observations (number of attributes).

5.2 Experimentation and Results

Figure 31 to Figure 35 show the goodness of the clusters for the projects. It reveals the numbers of clusters that are worth exploring for each particular project. The plots depict the overall sum of the sum of the distances from each input data point to its corresponding cluster. The dark circles enclose the clusters to explore having into account the fewer number of clusters and the lower result for goodness. They also show the quadratic regression line to show the tendency of the goodness, it is evident that for all projects the value of the goodness decreases as expected, nevertheless their decrease is worth analyzing. For projects C and D the decay is constant, but for project A the story is a little different, the goodness increases with fewer elements but decreases dramatically as the number of clusters increases, however it is possible to find a low value for 6 clusters, therefore that's a candidate in project A, and therefore worth exploring.

The shape of the quadratic regression in these plots reflect the morphology of the data to some extent, for instance, in project A the data could have naturally well defined clusters that are very disconnected from each other, so when the number of clusters increases the natural clusters are found and the distances for each cluster are diminished substantially, if the number of clusters increases even more, the FCM finds sub-clusters inside the natural groups, this may not be useful afterwards because the FCM has gone beyond the optimal level of granularity, which is the naturally defined clusters, also the natural clusters may have some well defined sub-clusters, this is reflected in the constant dramatic change in the goodness measure.

Project B shows a low value of goodness for 4 and 5 clusters; therefore they are selected as candidates; also the quadratic regression shows an interesting tendency that stabilizes if the number of clusters increases, this suggests that the data has natural clusters and that they are not very distant from each other, also the elements are homogeneously distributed inside their clusters, so if the FCM finds more sub-clusters the goodness may not change drastically.

As for projects C and D, the tendency of the goodness is constant, which means that the data may be equally spread out trough the space and that there are no natural clusters,

therefore the boundaries for the groups are not well defined. In project C there is a drastic change from 7 to 8 clusters, but clusters 5, 6, and 7 are almost constant, therefore they are worth exploring. In project D the goodness for 3, 5, and 6 clusters is very similar, which tells us that especially 3 clusters is a good candidate for project D, nevertheless 5, 6, and 7 clusters are also explored.

Finally for project E, there is a drastic change from 5 and 6 clusters to 7 clusters, therefore 7 groups is a good candidate for this data set, but also 5 and 6 clusters are explored.

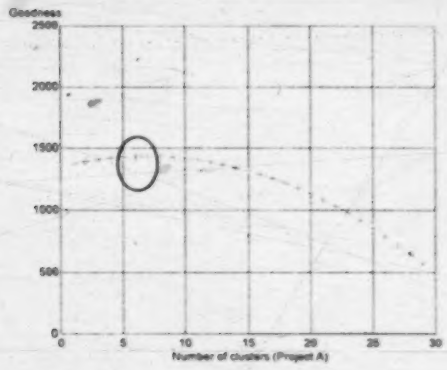


Figure 31: Clusters to analyze for project A

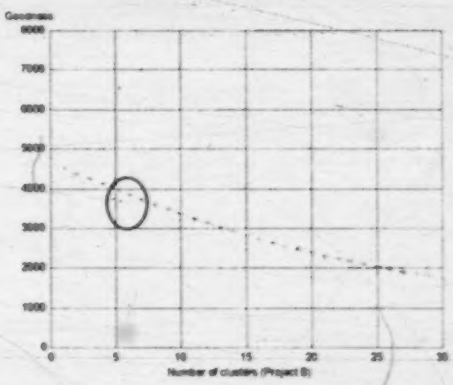


Figure 32: Clusters to analyze for project B

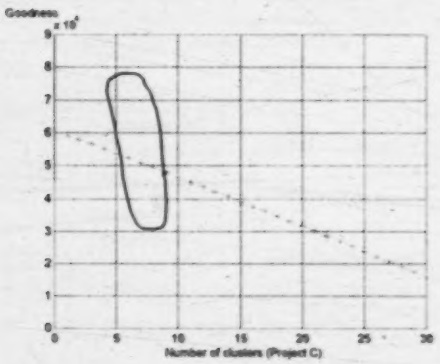


Figure 33: Clusters to analyze for project C

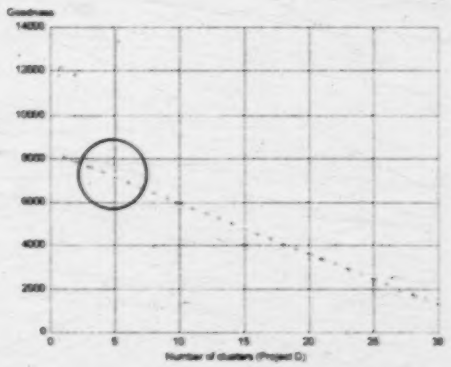


Figure 34: Clusters to analyze for project D

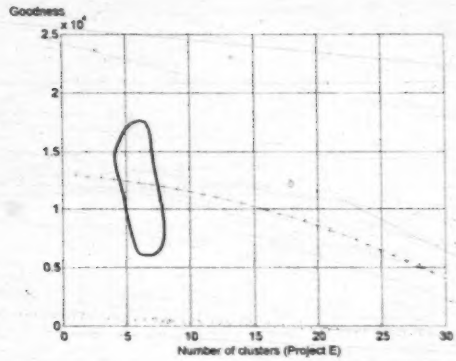


Figure 35: Clusters to analyze for project E

Now that the best number of clusters for each project has been found, they are explored using the local regression method and their performance is tested. The goodness of the clustering algorithm is calculated using Equation 14, then the regression models are estimated to obtain the slope "*s*" and offset "*b*" for the general linear equation $y = sx + b$. Each regression model represents one dimension of the data in each cluster. In this section the results obtained for project A are presented, the reader is encouraged to refer to Appendix C to find details about the results for the rest of the projects.

Lets remember that the data set of project A does not contain zeros in the output domain, it is not zero inflated, therefore the data is more spread out in the space than the data coming from the rest of the projects; for this reason, and for illustrative purposes, the data for project A is presented.

Table 9 shows the resultant coefficients of the regression models for project A considering 6 clusters.

	cluster 1		**cluster 2**		**cluster 3**		**cluster 4**		**cluster 5**		**cluster 6**	
Dimension	*slope*	*offset*	*slope*	*offset*	*slope*	*offset*	*slope*	*offset*	*slope*	*offset*	*slope*	*offset*
1	0.040	0.145	0.062	0.124	0.121	0.112	0.199	0.119	0.692	0.083	0.047	0.115
2	-0.246	0.322	-0.134	0.221	-0.028	0.159	-0.188	0.139	-0.204	0.167	-0.135	0.126
3	-0.266	0.158	-0.231	0.142	-0.238	0.145	-1.161	0.142	-0.267	0.167	-0.035	0.124
4	0.186	0.094	0.165	0.087	0.022	0.134	-0.232	0.147	0.380	0.128	-0.246	0.147
5	0.160	0.105	0.176	0.082	0.091	0.113	0.016	0.134	0.557	0.105	-0.056	0.132
6	-0.007	0.154	-0.036	0.143	0.168	0.118	0.501	0.131	0.825	0.131	0.214	0.116

Table 9: Regression coefficients for Project A considering 6 clusters

The information provided in Table 9 for cluster 1 (shaded area) is plotted in Figure 36 to Figure 41, they depict the data belonging to the cluster and its associated regression line in each dimension, each plot shows one dimension of the observations in the clusters. Since

the data is clustered using all 6 dimensions as inputs, the clusters are not very evident in the plots, there we are looking at each dimension at a time, so it is possible to get the impression that the data is overlapped, but let's not forget that we are looking at a single dimension in each plot. The plots in Figure 36 to Figure 41 are generated using 6 clusters and the normalized CK metrics data. The intension of normalizing the data is to prevent the dimensions with the highest values in their data to take advantage in the clustering algorithm; in this way it is guaranteed that each dimension has the same importance as the rest; also the number of fixes is normalized.

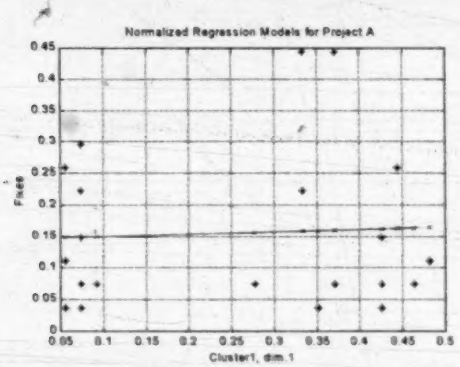


Figure 36: Model in cluster 1 for NoM

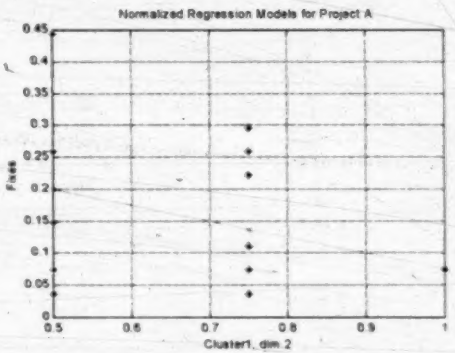


Figure 37: Model in cluster 1 for DIT

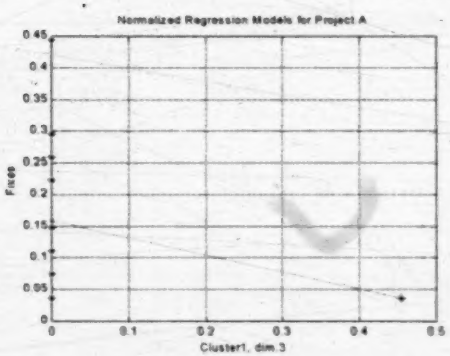


Figure 38: Model in cluster 1 for NOC

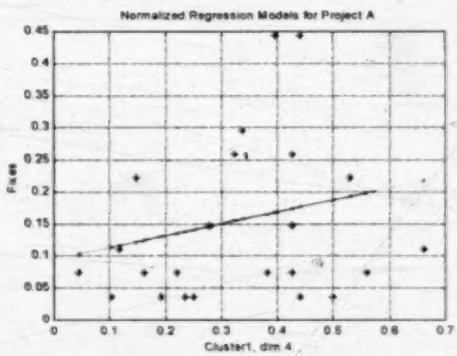


Figure 39: Model in cluster 1 for CBO

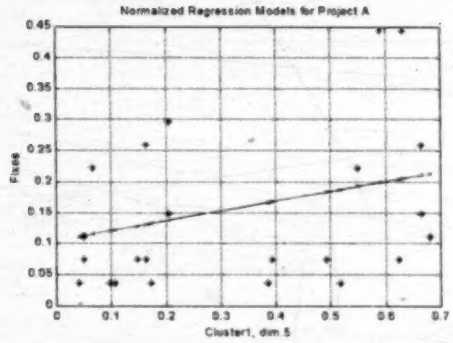


Figure 40: Model in cluster 1 for RFC

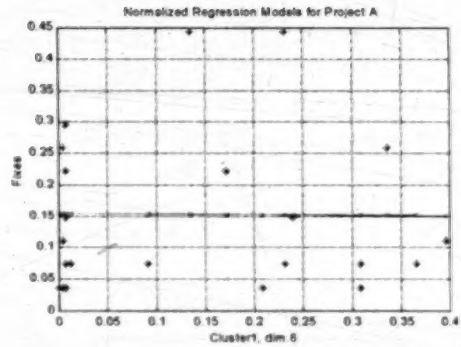


Figure 41: Model in cluster 1 for LCOM

It is to be expected to have the same number of data points in all plots since each plot is only a representation of one dimension of the same elements, nevertheless some plots show fewer components, such as Figure 37, this is because the data is overlapped and they ended up to be in the same position from this dimension perspective. This situation provides the regression algorithm with fewer data elements to build a linear equation, it also seems that the model has to cover different number of outputs for the same inputs, but in reality that's not the case. Figure 42 to Figure 45 show 6 regression models found in all clusters with the complete dataset, each plot demonstrate one dimension of the data points. Lets remember that the regression models are to be used only within the boundaries of their corresponding clustered data, nevertheless they are plotted in the complete range with the intension of illustrating their shape more explicitly.

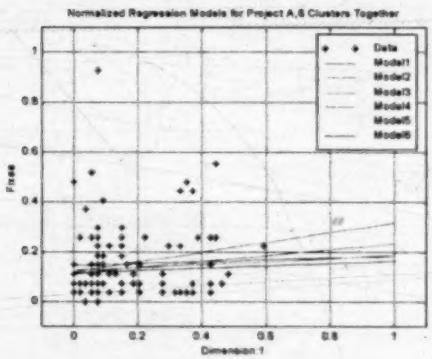


Figure 42: Linear models for NoM of project A

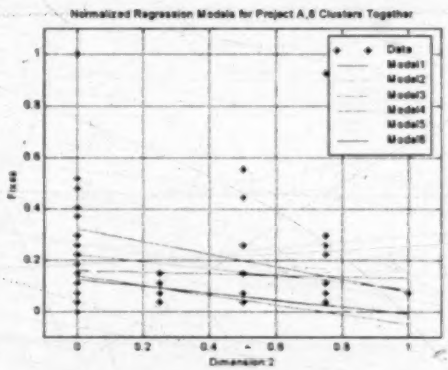


Figure 43: Linear models for DIT of project A

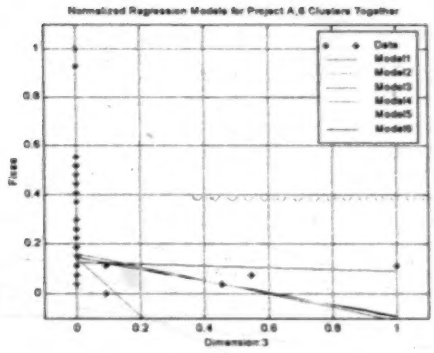


Figure 44: Linear models for NOC of project A

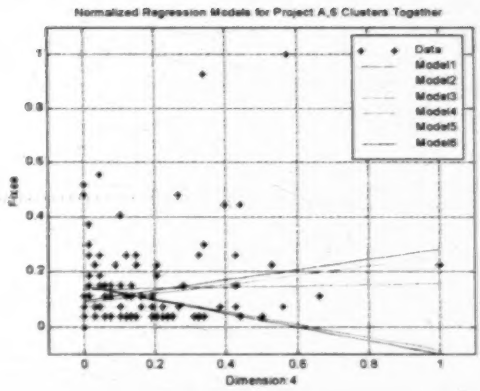


Figure 45: Linear models for CBO of project A

Figure 42 to Figure 45 show only the clearest dimensions of the data sets A. The goodness of the regression models is estimated by the mean squared error. Let y'_i be the prediction value of the regression line, y_i be the value to be predicted, and n the number of elements in the cluster.

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (y'_i - y_i)^2$$

Equation 16: Mean Squared Error

Table 10 presents the MSE values obtained for each regression model in each dimension. In the case of dimension 2 of cluster 1, the MSE value is around 10 times higher than the rest, this is because the data is overlapped from this dimension view and the data is far from the regression line, therefore the errors are added several times, in more precise terms, the number of times the data is overlapped.

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.043	0.048	0.047	0.017	0.020	0.024
2	0.322	0.314	0.306	0.032	0.059	0.025
3	0.044	0.040	0.043	0.030	0.060	0.042
4	0.056	0.060	0.068	0.022	0.041	0.019
5	0.072	0.079	0.084	0.020	0.028	0.027
6	0.034	0.034	0.030	0.028	0.028	0.018

Table 10: Error of the regression models in each cluster in each dimension

The final intension of the models is to be a tool to predict the number of fixes for the software classes, nevertheless there are 6 different models for the same clustered data, one for each dimension, but in some cases each model suggests a different number of fixes for

the same data point in the independent variable. Evidently this is a conflict for the models to accomplish their function; at the end there should be a single mathematical model able to provide the most accurate answer. In order to do so, a multivariate regression model is needed.

Unfortunately most of the input dimensions are linearly dependent between them, which means that they do not provide additional information for a multivariate model and only one of them is necessary to derive the model, this is the reason why the linear models are generated independently for each dimension at a time.

Table 11 shows the clusters that are useful and valid to build a multivariate model to predict the number of fixes. To be able to build a 6 dimension-multivariate model at least 8 data points are needed in the cluster, and the input dimension must not be linearly dependent. The clusters that are not shown in Table 11 are either linearly dependent in the input dimensions or do not have enough elements to build the multivariate model.

Valid clusters to build multivariate models for project A	**Valid clusters to build multivariate models for project B**	
For a total of 6 clusters	For a total of 5 clusters	For a total of 6 clusters
Cluster 1, cluster 4, cluster 6	Cluster 3, cluster 4	Cluster 1, cluster 4, cluster 6

Table 11: Valid clusters to build multivariate models

The following are the resultant multivariate regression coefficients for cluster 1 in project A, the resultant multivariate regression coefficients for the rest of the clusters are shown in.

Term	Coefficient
Intercept	-0.0640
Dimension1	-0.7276
Dimension2	0.2248
Dimension3	-0.2672
Dimension4	-0.0788
Dimension5	1.0605
Dimension6	-0.6385

Since the dimensions of the input data points correspond to the CK metrics, it is possible to rebuild the multivariate model depending on the normalized CK metrics to predict the normalized number of fixes, the following equation denotes the multivariate model for cluster 1 in Project A Therefore the model becomes:

$$fixes = -0.7276(NoM) + 0.2248(DIT) - 0.2672(NOC) - 0.0788(CBO) + 1.0605(RFC) - 0.6385(LCOM) - 0.0640$$

To appreciate how good this model predicts the normalized fixes Figure 46 presents the actual fixes and the predicted output y of the model. If the results follow a pattern of a 45-degree line then the prediction performs as desired, meaning that the predicted output y is matching the actual number of fixes. It is evident that the tendency of the distribution of the elements follows a 45-degree line; however there are some elements out of this tendency.

Figure 47 depicts the standardize residuals for the multivariate model. The residuals are the vertical distance between data points and the fitted multivariate regression line; positive

residuals indicate elements above the regression line while negative residuals are elements below the regression line. The Figure 47 plot shows how closely the computed regression line fits the variables. If the residuals form clusters above or below the line, the relation between the variables may not be linear. In this case, the elements are scattered in an area of +/- 2 standard deviations.

Figure 48 depicts the histogram for the frequency of the residuals with a superimpose normal distribution curve. If the residuals are normally distributed the curve should match the histogram well. In this case the histogram matches the normal curve in the outer areas but there are 2 bars out of the range, they represent elements in the range from 0.5 to 1 standard deviation, and from –0.5 to –1 standard deviation, therefore the data is not completely normally distributed.

The coefficient of determination R^2 is estimated to measure the how adequate the regression model is, where $0 \leq R^2 \leq 1$. R^2 is the amount of variability in the data explained by the regression model. Values of R^2 close to 1 indicate much variation in the output *y* has been accounted for by the predictors and the model can be considered as a good fit, whereas lower values of R^2 indicate a poor fit. In this particular case $R^2 = 0.5235$.

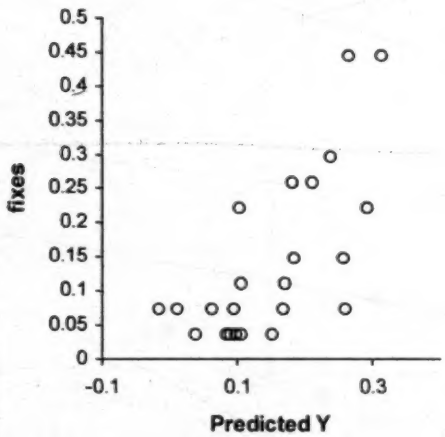


Figure 46: Normalized fixes Vs Predicted *y*

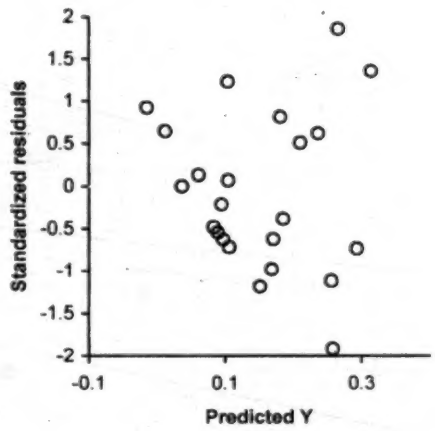


Figure 47: Standardize residuals

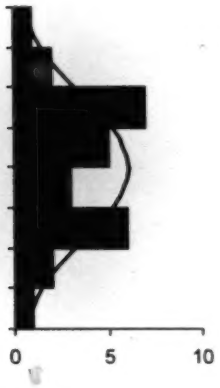


Figure 48: Histogram and distribution of residuals

5.3 Conclusions

The models to predict the fixes are built using multivariate regression lines when possible, and linear regression lines for each dimension for all cases. Most of the clusters examined contained attributes with linear dependency between them, which prevent us from building multivariate models for such clusters since the attributes do not provide additional information to the model.

When it is not possible to build a multivariate model, the single dimension models are to be used, nevertheless some models do not agree in the prediction of fixes in the same cluster, this is, the models from different dimension for the same cluster suggest different results even though they are linearly dependent. In such cases the model to use is the one with less error in the goodness test and higher coverage of the space, and the one with more elements in it, meaning that this attribute represents the cluster better than the others. To find the best model in each case the reader can refer to Appendix C. In most of the project the data is zero inflated, which makes the modeling hard to perform, also the findings suggest that the data is non-linear, meaning that another approach to model the data should be attempted, the use of neural networks is a candidate to this problem for its ability to model non-linear data.

Chapter 6. Neural Networks

This section presents the approaches explored to describe the software engineering data using neural networks (NN). The motivation to use NN to build a prediction system for the software data is that NN are powerful tools to model non-linear structures. This section describes several approaches using NN, each approach has a variation in terms of data or computational behavior. It is clear that using neural networks to predict the number of fixes will not give us a comprehensive mathematical model to analyze, however the system would be useful to provide the desired output in the prediction. The NN is a black box that receives some inputs (CK-metrics) and will produce the appropriate outputs (fixes).

This chapter describes five approaches to model the software data using neural networks. Since it is not possible to know how many neurons will provide the best neural network beforehand an iterative process of experimentation is performed, first a network of 2 neurons is built and tested, later the number of neurons in the hidden layer is increased by one and the process is repeated until a network of 45 neurons is built and tested; at the end, the results of all networks are compared and the configuration that provided the lowest error is reported. To keep uniformity, and with the intension of comparing the networks, the following parameters are kept constant throughout all approaches:

- *Learning rule to update the weights:* backpropagation
- *Topology of the network:* feedforward neural network
- *Number of hidden layers:* 1
- *Learning rate* = 0.0045
- *Model of the neurons in the hidden and output layer:* unipolar sigmoid function within the range [0-1].
- *Number of neurons in the hidden layer:* from 2 to 45
- *Number of neurons in the output layer:* 1
- *Training epochs:* 6000

The error is computed as the averaged sum of squared differences between the target and the NN output, according to Equation 16.

The approaches described in this chapter are:

Approach one (feed forward neural network): This approach concentrates on experimenting with a common neural network architecture. This approach serves as a reference to compare the behaviour of the rest of the approaches.

Approach two (Leave one out testing method): This method focuses on a different testing method, the training data set contains n-1 observations from the data set while the testing set comprises the remaining observation not included in the training set. The number of experiment to do for each particular network depends on the number of observations, each network configuration will be trained and tested n times, where in each experiment, the testing set consists of a new observation in the data set. Choosing the observations in the training and testing sets is an iterative process; the first experiment takes, as a training set, observations from 1 to n-1, keeping the remaining observation n as testing set. The second experiment takes observation n-1 as testing set and the rest as training set, the third experiment takes observation n-2 as testing set and the rest as training set, and so on until the testing set is observation 1 and the rest are used as training set.

Approach three (Constant weight modifiers): This approach introduces a constant weight in the learning process; the weight affects the derivatives of the backpropagation learning rule only when the training output is zero. This approach is suggested since the number of zero values in the target set is high; this method has the intention to minimize the effect of the zero inflation in the output domain.

Approach four (Constant weight modifiers and consistent elements): This approach focuses on the method described in approach three, but considering only the consistent element in the data sets, this is, all the identical observations in the input domain with different associated outputs are discarded.

Approach five (Constant weight modifiers and most correlated attributes): This approach takes into account only the four most correlated attributes to the number of fixes as data sets. The intension is to explore the behaviour of the networks without data that may be considered as noisy due to its low correlation to the target sets. The approach also takes advantage of the constant weight modifier in the learning process.

The following paragraphs are dedicated to give a brief background on the neural network theory, specially the feed forward with backpropagation learning rule. Latter the five approaches are described and commented.

6.1 Background on Backpropagation Neural Networks

The ability of neural networks to generalize, modify their behavior, and tolerance to noise in the input domain, accounts for much of the interest in using them as a modeling tool. The NN models have some important intrinsic properties, which are advantageous in the modeling context [27]. Some of these properties are: The distribution free property, which

allows construction of models without assumption about the underlying distributions of processes of interest; the learning capability, which allows the models to be constructed or adjusted solely based on the available data without the intervention of a programmer; and the parallel processing capabilities, which permit these models to be transferred to parallel hardware [27]. They are best suited for tasks requiring some type of pattern recognition in the input. Many different neural network architecture exist, each having their strength and limitations [14].

An artificial neural network can be defined by specifying the following characteristics:

- Model of neurons used
- Topology
- Learning rule to update the weights

6.1.1 Neuron model with unipolar sigmoid function

The Neural Networks comprise a set of interconnected neurons, each having a transformation function that it performs on the weighted sum of inputs to produce an output [24]. In other words, the Neural Networks are parallel systems inspired by the architecture of biological neural networks, comprising simple interconnected units (artificial neurons) [4].

Figure 49 shows the adjustable synaptic "weights" on the input lines to excite or inhibit incoming signals. An input vector $\mathbf{x} = (x_1, \ldots, x_M)$, considered to be a column matrix vector, is linearly combined with the weights $\mathbf{w} = (w_1, \ldots, x_M)$ via the dot product to form the sum $s = \sum_{m=1}^{M} w_m x_m = \mathbf{w}^t \mathbf{x}$.

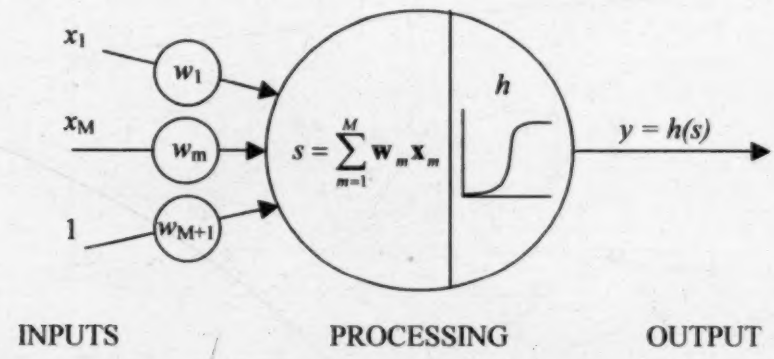


Figure 49: Neuron model with unipolar sigmoid function

In the previous figure, the activation function $y = f(s)$ maps a sum s into the proper range of output values, these discrete-valued functions have given away to continuously differentiable functions so that the gradient descent methods (explained later) can be used to solve for weights that map an input feature vector $\mathbf{x} = (x_1, \ldots, x_M)$ into the desired identifier (target) vector $y' = (y'_1, \ldots, y'_P)$ that represents a class.

Figure 50 presents the shape of the unipolar sigmoid activation function s; it is continuously differentiable and has the form:

$$y = \frac{1}{1 + e^{(-\alpha(s-b))}}$$

Equation 17: sigmoid function

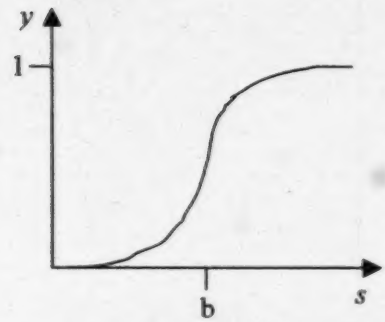


Figure 50: Sigmoid activation function

Where α is the decay (growth) rate, b is the *bias* that shifts the function center to where e^0 occurs (at $s = b$), where the output is midvalue $y = 0.5$. thus b is the s-axis center of asymmetry of $f(s)$.

There are several training algorithms that can be used to train the network, each having particular areas of specialty; the back propagation is the most common learning algorithm that has been used by software metrics researcher [4]. The back propagation algorithm is also to be used in this work.

6.1.2 Backpropagation learning rule with a single hidden layer

All models of artificial neurons try to resemble their biological counterparts as much as possible [16]. A biological neuron receives an electrical impulse through its dendrites, sums them up, and if the sum exceeds the neuron's body threshold it triggers an electrical signal along its axon. The strength of the incoming signals is determined by the synapse. This phenomenon is simulated in the artificial neural networks as a coefficient (weight) applied to the input signal coming from a specific dendrite.

A Multi Layer Perceptron (MLP) that has a single hidden layer is presented in Figure 51. There are M inputs branching nodes, P hidden neurons, and J output neurons. The weights of the input lines of the middle and output neurons are designated by $\{w_{mp}\}$

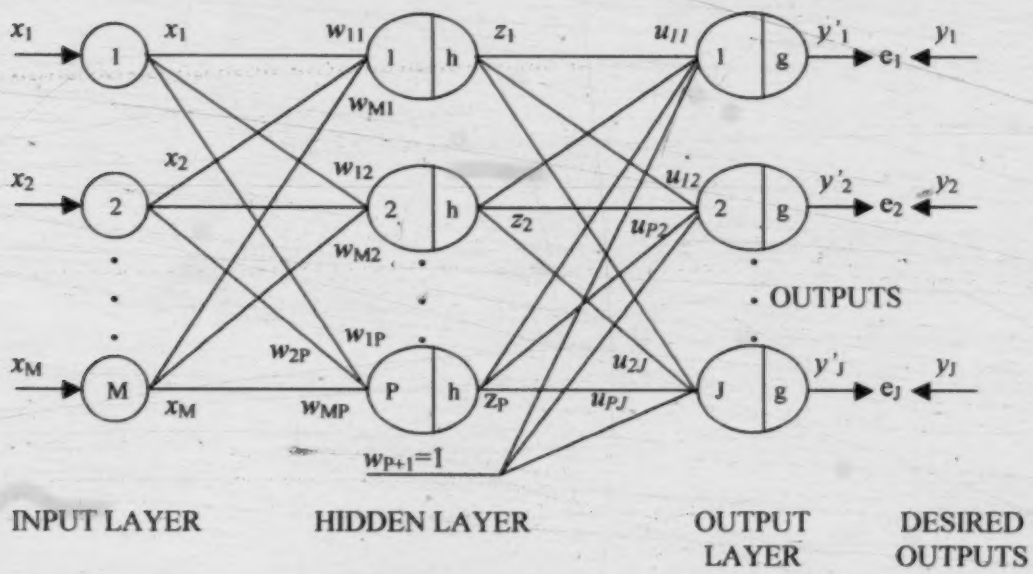


Figure 51: Feedforward neural network diagram

In Figure 51, both of the sigmoid functions at the hidden and output layer are of the same type, unipolar. The diagram does not show the offset for neurons in the hidden layer to prevent the figure to be unclear, but the neurons of the hidden layer have an offset as one of their inputs, $w_{M+1} = 1$.

Let there be a sample of Q observation vectors $\{\mathbf{x}^{(1)},\ldots, \mathbf{x}^{(Q)}\}$ from K classes, where $K \leq Q$. For each observation $\mathbf{x}^{(q)}$ there is an associated target output vector $\mathbf{y}^{(k)} = \mathbf{y}^{(k(q))}$ that identifies its class number $k = k(q)$. The task is to train the MLP by adjusting the weights $\mathbf{w} = (w_{11},\ldots, w_{MP})$ and $\mathbf{u} = (u_{11},\ldots, u_{PJ})$ as shown in Figure 51 until each observation $\mathbf{x}^{(q)}$ is mapped into an output $\mathbf{y}'^{(q)}$ that is very close to $\mathbf{y}^{(k)} = \mathbf{y}^{(k(q))}$.

To force each actual output $\mathbf{y}'^{(q)}$ towards the correct output $\mathbf{y}^{(k(q))}$, the weights are adjusted so as to minimize the *total sum-squared error* (TSSE) E between the targets and the actual outputs. The TSSE is calculated according to:

$$E = \sum_{q=1}^{Q} \left\| \mathbf{y}^{(q)} - \mathbf{y}'^{(q)} \right\|^2$$

Equation 18: Total sum-squared error (TSSE)

And the *total mean-squared error* (TMSE) is:

$$TMSE = \frac{1}{(QJ)} E$$

Equation 19: Total mean-squared error (TMSE)

$E = E(\boldsymbol{w}, \boldsymbol{u})$ is considered to be a function of the weights $\mathbf{w} = (w_{11}, \ldots, w_{MP})$ and $\mathbf{u} = (u_{11}, \ldots, u_{PJ})$, so E becomes:

$$E = \sum_{q=1}^{Q} \left(\sum_{j=1}^{J} \left[\mathbf{y}_j^{(q)} - g\left(\sum_{p=1}^{P} \mathbf{u}_{pj} h\left(\sum_{m=1}^{M} \mathbf{w}_{mp} \mathbf{x}_m^{(q)} \right) \right) \right]^2 \right)$$

Where $h(\cdot)$ and $g(\cdot)$ are sigmoid activation functions for the hidden and output layers respectively.

The function $E(\boldsymbol{w}, \boldsymbol{u})$ is a nonnegative continuously differentiable function on the weight space, which is $[-b, b]^{MP+PJ}$ $(b \geq 1)$, which is a finite-dimensional closed boundary domain that is complete and thus compact. Therefore, $E(\boldsymbol{w}, \boldsymbol{u})$ assumes its minimizing point $(\mathbf{w}^*, \mathbf{u}^*)$ in the weight domain. This does not mean that the sum-squared error E will be zero at the solution weight set $(\mathbf{w}^*, \mathbf{u}^*)$, but only E will assume its minimum value there on the given weight domain. If the target vectors $\{\boldsymbol{y}^{(k)}\}$ are chosen judiciously to be far apart and if the observations for different classes are not too close, then the minimum mapping will successfully recognize the input feature vectors by mapping them in their class identifiers.

To solve the minimizing weights set $(\mathbf{w}^*, \mathbf{u}^*)$, we use the necessary conditions:

$$\frac{\partial E(\mathbf{w}^*, \mathbf{u}^*)}{\partial w_{mp}} = 0$$

and

$$\frac{\partial E(\mathbf{w}^*, \mathbf{u}^*)}{\partial u_{mj}} = 0$$

we cannot solve these nonlinear equations in closed form, but we can approximate the solution $(\mathbf{w}^*, \mathbf{u}^*)$ iteratively with steepest descent.

To find a local minimum w_{locmin} for a nonlinear real valued function $y = f(w)$, we say $\frac{df(w)}{dw} = 0$ and solve for $w = w_{\text{locmin}}$. However, in general case of nonlinear functions we can only find an approximate solution w_{aprox} to w_{locmin} by iterative methods. Starting from

some initial point $w^{(0)}$, we move a step in the direction of steepest descent to $w^{(1)} = w^{(0)} - \frac{df(w^{(0)})}{dw}$, which is opposite to the direction of the steepest ascent. Lets note that the direction is either positive or negative along the w-axis. For the iterative $(r + 1)$st steep, we have:

$$w^{(r+1)} = w^{(r)} - \eta\left(\frac{df(w^{(r)})}{dw}\right)$$

The *step gain* $\eta > 0$ amplifies or attenuates the step size. If the step is too large, then it would move past the local minimum w_{locmin}. One way to define the value of η is by trial and error, until the network produces an acceptable result. The step gain is also called *learning rate*.

In general, a function $y = f(w_1, \ldots, w_M)$ of several variables can be locally minimized as:

$$(w_1, \ldots, w_M) \leftarrow (w_1, \ldots, w_M) - \eta\left(\frac{\partial f(w_1, \ldots, w_M)}{\partial w_1}, \ldots, \frac{\partial f(w_1, \ldots, w_M)}{\partial w_p}\right)$$

Equation 20: Estimation of w

In matrix form this is:

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \eta \nabla f\left(\mathbf{w}^{(r)}\right)$$

Equation 21: Estimation of $\mathbf{w}$ in matrix form

Where, $\mathbf{w} = (w_1, \ldots, w_M)$. The normalization of $\nabla f(\mathbf{w}^{(r)})$ to unit length would change η.

Now having the previous derivations in mind we can obtain the equations to compute the weights w of the hidden layer and the weights u of the output layer with unipolar sigmoid functions, the derivation of the equations is not specified in this chapter, but they can be found in [3]. The backpropagation learning rule with unipolar sigmoid functions can then be described as:

$$u_{ji} \leftarrow u_{ji} + \eta_1\left(y_j^{(q)} - y'_j{}^{(q)}\right) y'_j{}^{(q)}\left(1 - y'^{(q)}\right) z_p^{(q)}$$

Equation 22: Estimation of u (weights in output layer)

and

$$w_{mp} \leftarrow w_{mp} + \eta_2 \left\{ \sum_{j=1}^{J} \left(y_j^{(q)} - y'_j{}^{(q)} \right) \left[y'_j{}^{(q)} \left(1 - y'_j{}^{(q)} \right) \right] u_{pj} \right\} \left[z_p^{(q)} \left(1 - z_p^{(q)} \right) \right] x_m^{(q)}$$

Equation 23: Estimation of *w* (weights in hidden layer)

These equations are to be used in the five approaches described later in this chapter.

The following sections of this chapter provide a description of five different approaches to model the software data sets using artificial neural networks with backpropagation learning rule.

6.2 Approach one (feed forward neural network)

For the first study of the neural networks, the software data was randomly divided into 2 sets, 60% for training and 40% for testing. Since it is not possible to know how many neurons in the hidden layer will provide the best neural network beforehand an iterative process of experimentation is performed, first a network of 2 hidden neurons is built and tested, later the number of neurons in the hidden layer is increased by one and the process is repeated until a network of 45 neurons is built and tested; at the end, the results of all networks are compared and the configuration that provided the lowest error is reported. The learning follows the backpropagation rule of a feedforward neural network with two layers, one hidden layer and one output layer of a single neuron. All neurons in the network use a unipolar sigmoid function within the range [0-1]. The learning rate is set to 0.0045.

6.2.1 Experimentation and results

Table 12 shows a summary of the results obtained using neural networks, it comprises the information regarding the best number of neurons in the hidden layer and the corresponding Error.

Project	MSE	Hidden Neurons
A	0.2204	9
B	0.2349	6
C	0.1784	5
D	0.0943	26
E	0.1701	17

Table 12 characteristics of the best NNs in approach one

6.2.2 Conclusions

Unfortunately the plain neural networks did not provide high degree of accuracy, the prediction of the fixes is low according to the Error shown; and also the number of neurons is high in some projects. The NN may have problems to converge to a solution because there are too few observations in the set, and then when splitting the data into training and testing sets, they become even smaller, they seem not to have enough information to describe the complete morphology of the software. The Network is not finding the patterns that are hidden in the data. It is very possible that the data sets are incomplete; their inconsistency suggests such affirmation, there may be elements that are not present in the sets and that are necessary to show a complete picture of the patterns to the neural networks, the missing information may be in the form of extra attributes or in the form of missing observations.

We can do our best to minimize the effect of missing data in the sets to build a good prediction system, however the data sets sizes and the inconsistency of the data are important limitations. The fact that there are few elements in the sets suggest the need of another method for testing and training the networks, perhaps the use of the *leave x / N* testing method can help.

6.3 Approach two (Leave one out testing method)

For the second case study of the neural networks the conditions are very similar to those taking place in the first approach with the exception of the training and testing sets. The software data were divided into training and testing sets according to the leave 1 out method. The training data set contains *n*-1 observations from the data set while the testing set comprises the remaining observation not included in the training set. The number of experiment to do for each particular network depends on the number of observations, each network configuration will be trained and tested *n* times, where in each experiment, the testing set consists of a new observation in the data set. Choosing the observations in the training and testing sets is an iterative process; the first experiment takes, as a training set, observations from 1 to *n*-1, keeping the remaining observation *n* as testing set. The second experiment takes observation *n*-1 as testing set and the rest as training set, the third experiment takes observation *n*-2 as testing set and the rest as training set, and so on until the testing set is observation 1 and the rest are used as training set. The testing is done over all the elements of the data sets. The numbers of hidden neurons in each network go from 2 to 45. The Error between the NN output and the testing data set is computed, and the configuration of the network with the lowest error is reported. The learning follows the backpropagation rule of a feedforward neural network with two layers, one hidden layer and one output layer of a single neuron. All neurons in the network use a unipolar sigmoid function within the range [0-1]. As before, the learning rate is set to 0.0045.

Cross-validation can be used simply to estimate the generalization error of a given model, or it can be used for model selection by choosing one of several models that has the smallest estimated generalization error. For example, one might use cross-validation to choose the number of hidden neurons. The leave-one-out cross-validation often works well for estimating generalization error for continuous error functions such as the mean squared error [30]. To estimate the generalized error we use Equation 16.

6.3.1 Experimentation and results

Table 13 shows the generalized error for each project and the number of hidden neurons in the network that produces the lowest error.

Project	MSE	Hidden Neurons
A	0.2146	8
B	0.2224	6
C	0.1817	6
D	0.0900	27
E	0.1706	16

Table 13: characteristics of the best NNs in approach two

6.3.2 Conclusions

In general, the leave one out testing method does not improve the results with respect to the normal approach of dividing the data sets into 60% for training and 40% for testing, as shown in approach one. The generalized error and the number of neurons in the hidden layer are very similar to the results shown in approach one. Besides the lack of improvement in the results, the model to choose using the leave one out method is somehow ambiguous, the number of neurons shown in Table 13 correspond to the network that provided the lowest error from the *n* experiments in each data set. Such configuration of the network is not necessarily the best for all the observations in the data set.

In the leave one out testing method there are *n* experiments for each data set, which makes the computational effort much higher that with the normal testing method of splitting the data into 60% for training and 40% for testing. Based in this fact and on the lack of improvement in the results with this approach, the leave one out testing method is discarded from the experimentation in this work. From now on the data sets are divided into 60% for training and 40% for testing, in fact, the training and testing data sets to use in the rest of the approaches are the same as those used in approach 1 so the different approaches can be compared.

6.4 Approach three (Constant weight modifiers)

It has been mentioned before that the data is inconsistent in most of the projects; for instance, there are different outputs for the same set of inputs, in some cases these differences are in the order of $1x10^2$. Also the data is zero inflated in the output domain, and the sets contain at most 120 elements, which makes the set small for the neural networks to learn and to train from them. The new approach is to minimize the effect of the zero data in the output domain, to do so, the derivative equations for the NN are modified to distinguish from the zero and non-zero data. This approach introduces a constant weight in the learning process; the weight affects the derivatives of the backpropagation learning rule only when the training output is zero. A constant weight is added in the derivative equations to allow the learning when the output data is non-zero, and to restrict it when the data is equal to zero. This method has the intention to minimize the effect of the zero inflation in the output domain.

6.4.1 Experimentation and results

The new equation to calculate the weights in the hidden layer is:

$$w_{mp} \leftarrow w_{mp} + C\eta_2 \left\{ \sum_{j=1}^{J} \left(y_j^{(q)} - y'_j{}^{(q)} \right) \left[y'_j{}^{(q)} \left(1 - y'_j{}^{(q)} \right) \right] \mu_{pj} \right\} \left[z_p^{(q)} \left(1 - z_p^{(q)} \right) \right] x_m^{(q)}$$

Equation 24: Estimation of w (weights in hidden layer) with constant weight modifier

Where C is the constant weight modifier.

Please note that the constant weight modifier C affects the learning of the network, if the value of C is close to 1, then the learning is not affected, and the learning rate is kept the same; but if the value of C is different than 1, the learning is affected making the step gain η bigger or smaller. In this research the value of C goes in the range [0.1, 1.0]. When the target value y is zero, the value C is kept as 1, this leaves the learning of the network in the usual way; however, when the target value is different than zero, C is changed to a value smaller than 1, making the step gain smaller and allowing the network to learn in more detail. In this way the network performs in the usual way when the target value y is zero and tunes the learning finer when the target value y is different than zero.

The networks are built and tested for the values of C in the range [0.1, 1.0], therefore there are 10 different results for a particular network configuration. The network configuration is also changed to have different number of neurons in the hidden layer. At the end the results are compared and discussed. The behavior of the NN is explored using different number of hidden neurons and different values in the constant weight modifiers: $2 \leq$ *hidden neurons* ≤ 45, and $0.1 \leq C \leq 1.0$. For each experiment there is a combination of hidden neurons and value of C, therefore this approach consists of 430 experiments for each data set, 43

different neurons with 10 different constant weights each. These set of experiments cover all possible situations in the use of the NN with the data sets, but everything has a price and in this case it's the computation required for this approach. Since the results of approach two suggest that the leave one out testing method does not improve the results of the networks significantly, in this approach the data sets are divided as 60% for training and 40% for testing, in fact the sets are the same as those in approach one.

Table 14 presents the characteristics of the NN that best modeled the data sets, it shows the error between the testing data set and the NN output, the number of hidden neurons used to produce these results, the weights used to obtain the lowest error in each network.

Project	MSE	Hidden Neurons	*C*
A	0.2194	9	1.0
B	0.1206	7	0.6
C	0.1405	5	0.8
D	0.0981	26	1.0
E	0.1672	16	1.0

Table 14 characteristics of the best NNs in approach three

6.4.2 Conclusions

In most cases the weighted method proved to be better than the simple NN described in approach one; however the results depend greatly on the number of zeros the data sets have at the output domain. If the data has many zeros in the output domain, the NN is left only with few elements to tune the learning of the data. If the data set is too small then this method may fail in improving the accuracy of the models.

Since the data in project A does not contain zeroes in the output domain, the weighted method is not expected to produce better results for it, nevertheless the method was also tested in this set to explore its behavior and to build a comparison table. The error of project A is very similar to that shown in approach one, as expected, and the number of neurons to produce such an error is exactly the same.

It is interesting to see that the value of *C* for projects D and E is one, these sets are small, suggesting that there are not enough observations with non zero targets to characterize the data set and build a model using the constant weight modifier, however the results are not so different to those shown in approach one, in these two sets this method proved useless.

From the results it becomes evident that the weighted method performs better when the data is zero inflated in the output domain, in almost all cases the results are improved. Whoever the prediction of the data does not depend only on the percentage of zeroes forming the data set, but also on the morphology and distribution of the elements in the

space, lets also remember that the data is inconsistent in some projects having different outputs for the same set of inputs. Having this in mind the improvement of the NN using the weighted method is clear.

6.5 Approach four (Constant weight modifiers and consistent elements)

The fourth approach to model the software engineering data is very similar to the third approach, the weighted method (constant weight modifiers) applied to the NN is also used in this approach, with the only difference that the data now does not contain the inconsistent elements, the elements in the input domain that are the same but have different outputs are discarded. Since these data elements have different outputs it is not possible to know which element is the correct one, therefore all of them are discarded in order to reduce the amount of noisy data from the sets. It is expected that the NN does a better job since it does not contain inconsistent observation, but lets not forget that now the data sets are even smaller than they were before, this situation may also complicate the job of the NN to extract the essence of the data.

6.5.1 Experimentation and results

Table 15 shows the lowest error for each project, the weight, and number of hidden neurons for each network.

Project	MSE	Hidden Neurons	Weight
A	0.1092	6	1.0
B	0.1178	6	0.4
C	0.1454	5	0.8
D	0.0992	26	1.0
E	0.1673	16	1.0

Table 15 characteristics of the best NNs in approach four

6.5.2 Conclusions

In this approach, it is possible that by discarding the non-consistent observations, also some valuable elements are discarded; in some cases the distance between the different targets y is not too high, and therefore these data provided information about the morphology of the sets more than introducing noise. It is interesting to see that the error and number of neurons increase considerably in project A, probably some important observations were taken away, and even though they seemed to be inconsistent, they provided useful information to the neural network, now the network requires even more neurons to map the data set. As for projects C, D, and E, the error decreases but the number of neurons and constant weights are the same, this indicates that the elements that were discarded were, in fact, introducing noise to the data sets. These three sets are clear examples that the these sets have inconsistent elements that should not be there.

6.6 Approach five (Constant weight modifiers and most correlated attributes)

The fifth approach to model the software engineering data is similar to the third approach, the constant weight modifier is introduced to the NN to alter the learning process when the target in the training set is zero, but this approach takes into account only the four most correlated attributes to the number of fixes as input data. The intension is to explore the behaviour of the networks without data that may be considered as noisy due to its low correlation to the target sets.

6.6.1 Experimentation and results

Some of the input attributes are not correlated to the output data as illustrated in Figure 52 to Figure 56, therefore these attributes may not reveal the morphology of the data in its true sense. Having this in mind the hypothesis that these attributes can be noisy data or corrupted attributes can take place, and they may be discarded.

Figure 52 to Figure 56 present the correlation between all the CK metrics and the number of fixes. From these plots it is possible to appreciate and select the attributes to use as inputs to the NN. In most of the cases the correlation is below 30%, and considering that the sizes of the data sets are small, the NN may have some difficulties to find an appropriate model to map the entire space.

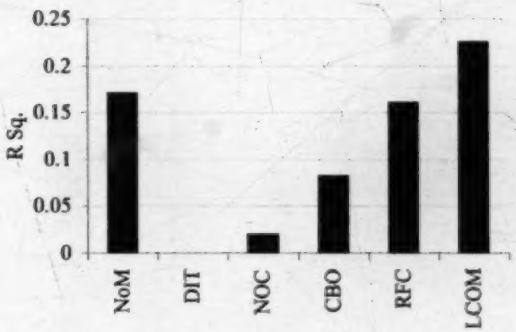


Figure 52: R^2 of attributes to fixes in project A

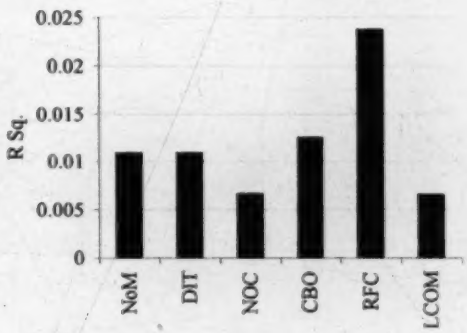


Figure 53: R^2 of attributes to fixes in project B

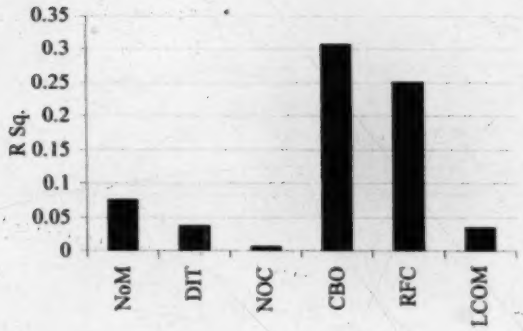


Figure 54: R^2 of attributes to fixes in project C

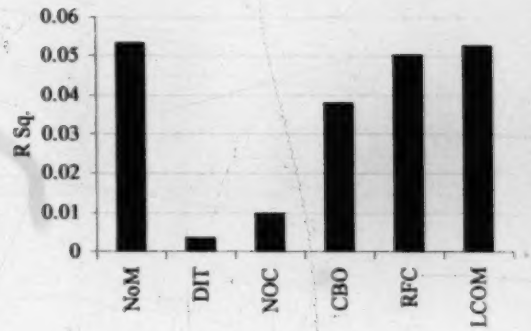


Figure 55: R^2 of attributes to fixes in project D

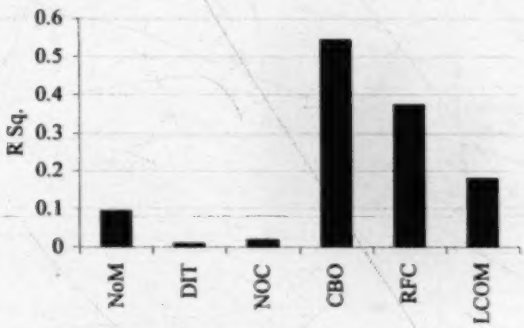


Figure 56: R^2 of attributes to fixes in project E

The attributes to train the neural networks were selected based on the information provided in Figure 52 to Figure 56; again all the possibilities of number of neurons and constant weight modifiers are taken into account to produce the results. The following table shows the outcome of the NN behavior using the best-correlated attributes to the number of fixes. It presents the attributes used as input data, the lowest error, the number of hidden neurons in the networks to produce such lowest error, and the constant weight modifiers for such networks.

Project	Used Attributes	MSE	Hidden neurons	Weight
A	NoM, CBO, RFC, LCOM	0.1109	9	1.0
B	NoM, DIT, CBO, RFC	0.1190	8	0.6
C	NoM, CBO, RFC, LCOM	0.1611	8	0.5
D	NoM, CBO, RFC, LCOM	0.0992	24	1.0
E	NoM, CBO, RFC, LCOM	0.1661	26	0.9

Table 16 characteristics of the best NNs in approach five

6.6.2 Conclusions

It is interesting to note that attribute NOC is never present in any of the best-correlated data sets and the attribute DIT is present only in project B. the hypothesis suggested at the beginning of this approach does not work for projects B and C but it works for projects A, D, and E. Based on the results we can be sure that the attribute NOC is not providing useful information to the NN to predict the number of fixes, this attribute is probably noisy and inconsistent.

Table 16 shows that the error is significantly smaller than those presented in approach three in most of the projects, this suggests that some attributes may be very noisy and do not provide useful information to the models, these attributes are not related to the number of fixes in the R^2 sense. In fact, this approach suggests that the best solutions are obtained when appropriate attributes are chosen to build the models; by doing so, it is feasible to obtain good results instead of by choosing complex method of building models.

The performance of the neural networks in this approach is also affected by the fact that the data sets are small, the original data sets have at most 120 elements, and since the non-consistent data was discarded in this approach, then the data sets does not contain enough information for the NN to train. Also the data points are spread out trough out the space,

therefore the neural networks have an even harder job to construct the hyper planes and cover the entire space.

6.7 Discussion

This chapter describes the behaviour of the neural networks in modeling the software engineering data sets. Five approaches are described:

Approach one: Classical neural network with back propagation learning rule.

Approach two: Classical neural network with leave one out testing method.

Approach three: NN with constant weights modifiers in the learning function.

Approach four: NN with constant weights modifiers in the learning function, and consistent observations only.

Approach five: NN with constant weights modifiers in the learning function, considering only the 4 most correlated attributes to fixes.

Figure 57 presents a comparison chart of the behaviour of the neural networks for all five approaches. It shows the averaged squared error between the network's output and the actual number of fixes for each specific project.

As for approaches one and two, there is not an obvious difference in the learning of the neural networks, for projects A, B, and D it shows some improvement but the performance is comparable to that obtained in approach one. These results suggest that the leave one out testing method does not really help or enhance the neural networks behaviour; nonetheless the leave one out testing method requires much more computation, making this approach difficult to compute. For such reasons the leave one out testing method is discarded and from now on the data sets are divided into 60% for training and 40% for testing.

In general, approaches thee, four, and five provide the best results, these approaches have in common the constant weight modifiers in the learning functions. However, as expected, approach three does not have a significant impact on the learning of the network in project A, lets remember that this project does not contain any zeros in the output domain, which makes the constant weight modifiers jobless for this data set. However there is a significant improvement in the behaviour of the approaches that use the constant weight modifiers in project B and C, it does not prove to be effective in projects D and E. it is interesting that the results for project D are not significantly enhanced by introduction of the constant weights in the learning functions because such project has 70.45% of zeros as number of fixes, yet it has very few observations in it, only 44; this results suggest that such project does not have enough information to describe the data sets accurately.

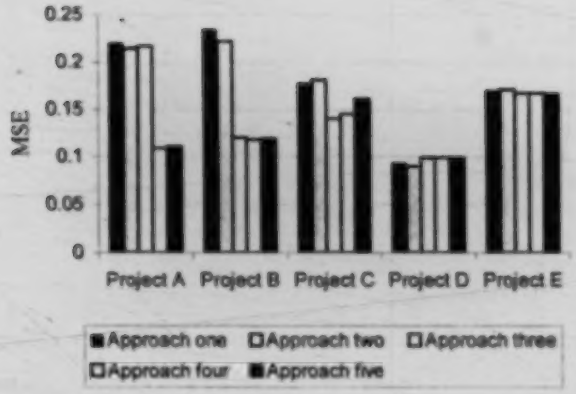


Figure 57: Summary of the NN behaviour

Table 17 presents the number of neurons that the neural network used to provide the lowest error. It shows that the network of projects D and E require a high number of neurons to provide accurate results; this could be a result of the dispersion of the observations in the multidimensional hyper plane and the fact that the sizes of such data sets are small, 44 and 38 elements only. It is also interesting to see that the number of neurons required for the networks in approach five is the highest compared to the rest of the approaches in all projects but D, situation that suggests that the attributes taken away from the data sets do provide useful information to the models even when they are less correlated to the number of fixes; such hypothesis can be confirmed by looking at the errors of approach five in Figure 57, where the error of the networks is similar to those of approach four or three. The introduction of the constant weight modifiers does not seem to make an impact in the number of neurons required for the neural networks, the networks in approaches three and four have similar number of neurons than those in approaches one and two, this indicates that the training of a neural network with constant weight modifiers may not be affected in terms of computational effort.

Approach	Project A	Project B	Project C	Project D	Project E
Approach one	9	6	5	26	17
Approach two	8	6	6	27	16
Approach three	6	7	5	26	13
Approach four	6	6	5	26	13
Approach five	26	8	8	24	26

Table 17: Best number of neurons for each data set

One characteristic that has to be mentioned for approaches three, four, and five is the value of the constant weight modifiers of the learning functions in the networks. Table 18 presents the values of the constant weight modifiers of the neural networks in each project, since approaches one and two do not use a constant weight modifier in the learning

functions, their values are shown as 1, such value does not make any influence in the learning of the networks.

Approach	Project A	Project B	Project C	Project D	Project E
Approach one	1	1	1	1	1
Approach two	1	1	1	1	1
Approach three	1.0	0.6	0.8	1.0	1.0
Approach four	1.0	0.4	0.8	1.0	1.0
Approach five	1.0	0.6	0.5	1.0	0.9

Table 18: Constant weight modifiers in each data set

It is interesting to see that the value of the modifiers in projects three and four in projects D and E is close to 1, the constant weights are not making any impact on the learning of the networks, and the lowest error is acquired. Also the number of neurons is exactly the same and the error is almost identical for such cases, in these examples the weights do not provide any help to the neural networks to improve the results, this may happen because the data sets are small and may not provide enough information as to predict the number of changes. However for projects B and C, the weights play an important role in the learning of the networks, as seen in Table 18 and Figure 57, where the error decreases with the introduction of the constant weight modifiers. In project A, the weights should not affect the learning of the neural networks because there are not zeros in the number of fixes.

At the end it is not possible to derive a unique model that can represent all data sets, or an approach that is the optimal for all data sets. In general we can say that the constant weight modifiers improve the learning of the networks as long as the data set is big enough or has enough observations to represent the project. It is revealed that the errors decrease with the implementation of such weights, but they are only useful when the data is zero inflated.

Chapter 7. Switching Regression Models and Fuzzy Clustering

The algorithm described in this section to build prediction models is explained in full detail in [23], from where the idea is borrowed. In this particular case, this algorithm is applied to the software data using one dimension at a time and the number of fixes, obtaining at the end a linear regression model to represent each dimension.

The motivation of using this algorithm with the software data is that, as mentioned before, the data inputs have different associated outputs; therefore using an algorithm capable of providing more than one output is useful for an accurate modeling, and this algorithm has such capability.

7.1 Background on Regression Models and Fuzzy Clustering

Let $S = \{(\mathbf{X}_1, y_1), \cdots, (\mathbf{X}_n, y_n)\}$ be a set of data where each independent observation $\mathbf{X}_k \in \mathbf{R}^s$ has a corresponding dependent observation $y_k \in \mathbf{R}$. Usually, the search for a best function is partially constrained by choosing the functional form of f in the assumed relationship

$$y = f(\mathbf{X}; \boldsymbol{\beta}) + \varepsilon$$

Equation 25: Form of the regression model

Where $\boldsymbol{\beta} \in \Omega \subset \mathbf{R}^s$ is a vector of parameters to be determined, ε is a random vector with mean vector $\mu = o \in \mathbf{R}^t$, and Ω is a set of feasible values of $\boldsymbol{\beta}$ [23].

Generalizing the idea, it is possible to redefine Equation 25 to represent the set of solutions for the clusters, therefore:

$$y = f_i(\mathbf{X}; \boldsymbol{\beta}_i) + \varepsilon_i, 1 \le i \le c$$

Equation 26: Generalized form of the regression model

Where $c \in [1,2,\cdots N]$ and N is the number of clusters.

For this particular project we assumed the form of Equation 25 to be a linear equation, and as for the clustering algorithm, the partition matrix u is such that $\sum_{i=1}^{c} u_i = 1$.

The general algorithm in this method is as follows:

1. Given data $S = \{(\mathbf{X}_1, y_1), \cdots, (\mathbf{X}_n, y_n)\}$. Specify regression models of the form of Equation 26, and choose a measure of error $E = \{E_{ik}\}$ so that $E_{ik}(\beta_i) \geq 0$ for i and k. Pick a termination threshold $\varepsilon > 0$.

2. Calculate values for the c model parameters β_i that globally minimize over Ω.

3. Update the partition matrix u.

4. Check for termination condition; otherwise go back to step 2.

For further details on this algorithm, the reader is encouraged to refer to [88].

The sum of errors is estimated to measure the goodness of the regression models according to Equation 27. Let **X** represent the data elements in the space formed by each dimension of the CK metrics and their corresponding fixes, and V the cluster prototype regression model; then:

$$error = \sum_{j=1}^{c} \sum_{i=1}^{n} \left\| X_{ji} - V_j \right\|^2$$

Equation 27: Error of the Regression Models and Fuzzy Clustering

Where c is the number of clusters and n the number of elements in the jth cluster.

7.2 Experimentation and Results

This method is first testes with artificial data set to appreciate its sensibility and behavior, later the method is used to cluster and model the software data sets. In both cases the error of the regression models is computed and compared.

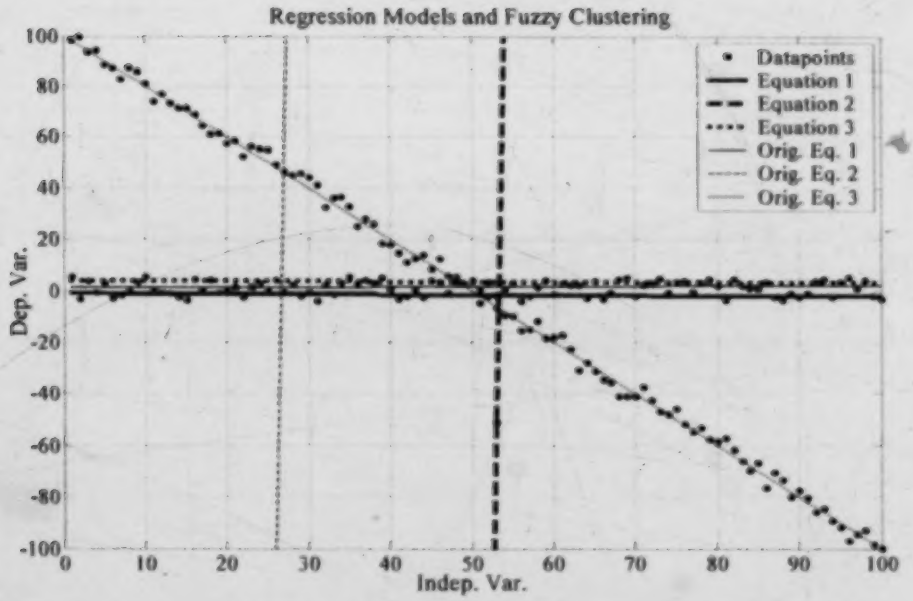


Figure 58: Sensitivity of the switching regression models and fuzzy clustering

It shows that for equation 3 in the plot, the initial position of the regression model is close to an obvious pattern in the artificial data, but the final regression models is covering data elements very close to those belonging to equation 1. Also notice that equation 2 is moved to the right side of the space however this cluster may not be needed according to the morphology of this data set, but it is interfering with the behavior of the equation 2, making it find incorrect elements to represent.

Table 19 provides the parameters *offset* and *slope* for the regression models and the error found in each cluster of project A. Table 19 considers 2 clusters for each dimension. The results for the rest of the projects and for different number of clusters can be found in Appendix D.

Dimension	Cluster	Slope	Offset	Error
1 (NoM)	1	-0.3680	23.4119	0.0336
	2	0.1032	0.4685	0.0465
2 (DIT)	1	5.7144	0.2429	0.4028
	2	-6.0384	14.8461	0.3588
3 (NOC)	1	-0.1814	0.4990	0.0633
	2	-4.7722	11.2871	0.0447
4 (CBO)	1	0	22	0.0377
	2	0.1619	0.6304	0.0601
5 (RFC)	1	0.0110	1.9794	0.0656
	2	-0.2622	70.5245	0.0695
6 (LCOM)	1	0.0038	1.2331	0.0121
	2	0.9467	1.5769	0.0287

Table 19: Regression models and errors for each cluster in project B

Figure 59 presents a visual example of the regression models found for dimension 6 (LCOM) of project B considering 2 clusters. The thin lines depict the original position of the regression models, and the thick lines the final regression models after the clustering. In this particular case, the regression models 2 did not change much, but as for the regression model 1 the readjustment is dramatic. The model 1 covers the points at the left side of the dimension and the model 2 the rest of the data.

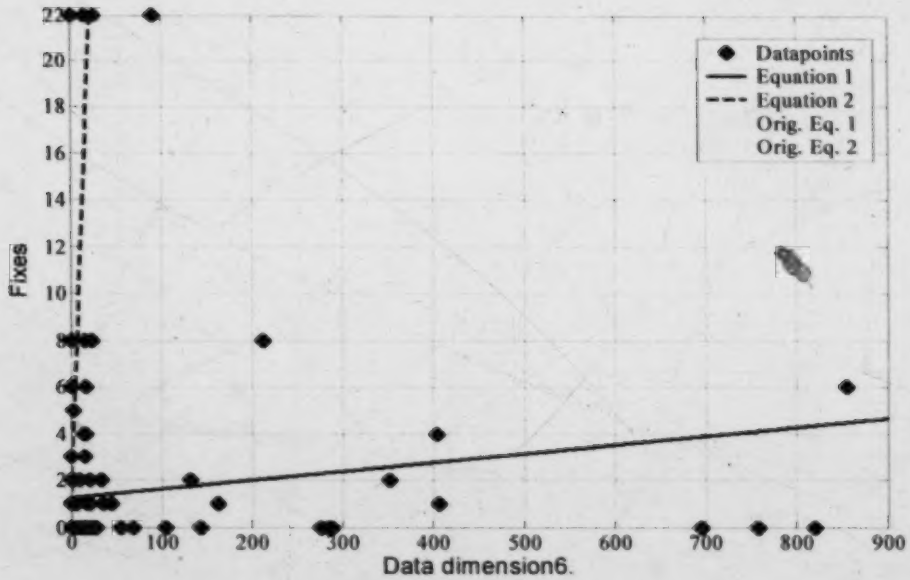


Figure 59: Regression models for project B, view from 6th dimension

7.3 Conclusions

Unfortunately this algorithm is not very robust when dealing with many regression models, after 3 clusters the prototypes do not follow any particular shape the data may have. Also the initial position of the prototypes is an important factor to consider to build the models, it can happen that the algorithm does not find any particular pattern if the starting regression model is not close to the appropriate elements in the set. For these reasons the algorithm is used to obtain 2 clusters in each dimension of the data set. Figure 59 is presented as an example of the sensitivity of the algorithm using artificial data.

The switching regression models and fuzzy clustering is a good tool when dealing with data that contain different outputs for the same inputs. The software engineering data used to build the models falls into such category but is not a unique characteristic of the data sets.

Unfortunately it is found that this algorithm is not very robust when it tries to find several clusters in the data set, the regression models become unorganized and inaccurate, with larger data sets the algorithm may provide better results because there would be more data to cover in the space.

This algorithm is also sensitive to the starting position of the regression models or prototypes, they do not end up covering obvious shapes unless they are positioned close to

such patterns from the very beginning of the algorithm. This is definitely a limitation when finding multiple patterns or models that are not obviously organized in the data sets.

For the particular case of the CK metrics data sets the algorithm provided useful models for each dimension, the error is in general comparable in magnitude to the error obtained in the clustering and local regression algorithm, however the switching regression models and fuzzy clustering is sensitive to the initial position of the regression models.

Lets also remember that this method is applied to each dimension at a time, therefore the elements in each dimension may be overlapped in several projects, this situation produces less data points from where to obtain accurate regression models, so when finding several clusters it becomes difficult for the algorithm to find such shapes, leading to obtaining poor results in such case.

Chapter 8. Genetic Algorithm-Based Clustering Method

So far the data has been analyzed to check its consistency according to its attributes, for instance, the R^2 between the input attributes and the number of fixes has been obtain to gain knowledge regarding their relationship, the CK metrics have been analyzed for each dimension and have been plotted against the output domain to visualize their morphology, and the FCM clustering has been used to separate the data into different classes. It has been shown that some of the input attributes are not related to the output elements and that they can be considered as noisy or inconsistent data; as well, some input elements have inconsistent associated output data, however the results suggest that they provide useful information up to some degree.

Lets undertake a different point of view now, lets explore the possibility that the data may be is incomplete; probably the data sets do not have all input attributes as they should, this thinking brings a new perspective of the data sets and their modeling. The questions that arise now are: what data is missing from the data sets? Could there be some missing attributes that could make the data consistent? How to find the missing data? Would the missing data make the modeling algorithms more efficient? These questions provide a new set of possibilities to explore, new challenges and more excitement into the research.

The first part of this chapter provides the background of this method, the second section gives a general description of Genetic Algorithms (GA), later the chapter focuses on experimentation and testing of this method with artificial data sets to validate its performance, and finally the method is used with the datasets of the five projects for this thesis (A, B, C, D, and E).

8.1 Background on Genetic Algorithm-Based Clustering Method

This chapter presents a method that creates a consistent dataset and an appropriate model from an inconsistent dataset. The method actually looks for missing information in the original dataset and expands the set of input variables with some auxiliary variables using the help of the Genetic Algorithm (GA). At the same time, this method builds a model that best fits the data taking into account the new proposed auxiliary variables. At the end the method provides the missing information in the dataset as well as the best model for it.

The new data can be used as a baseline to find the missing information in the real phenomena by purely knowing its characteristics, or it can be used as part of the information that cannot be obtained in experimental way because it is simply not available.

When the researcher know what kind of information is missing then the task of finding it from the phenomenon's source becomes easier, the information provided by this method can serve as a guide to know the characteristics of the data that is not present in the records, and can give a hint on where to look for it.

It is essential to define inconsistency in a given dataset before any attempt is made to make it consistent, this will provide the baseline and a starting point for the method described in this chapter. Lets start by defining the concept of inconsistency in a given data set; first we consider a consistent dataset $D = \{(\mathbf{X}_1, y_1), \cdots (\mathbf{X}_i, y_i) \cdots (\mathbf{X}_n, y_n)\}$, where each independent observation $\mathbf{X}_i \in \mathbf{R}^m$ has its corresponding dependant observation $y_i \in \mathbf{R}$. We can assume that a relationship between the independent observations (inputs) $\mathbf{X}_i$ and their associated dependant values (outputs) y_i exists for all i. Such relationship can be expressed as a function of the inputs $\mathbf{X}_i; 1 \leq i \leq n$, and some parameters $\boldsymbol{\beta} \in \varphi \subset \mathbf{R}^{m+1}$. Therefore,

$$y_i = f(\mathbf{X}_i; \boldsymbol{\beta}) + \varepsilon_i$$

Equation 28: General model.

where φ is a set of applicable values for the parameters $\boldsymbol{\beta}$ in the function, and ε_i is the noise for each observation. The values for each element $\boldsymbol{\beta}_m$ can be estimated using any regression method, for example, the Least Squared Error (LSE) if y is described by a linear relationship.

Lets now compute the distances from each observation $\mathbf{X}_i$ to the rest of the independent elements in the dataset D, so that $\Delta x_k = \left\| \mathbf{X}_i - \mathbf{X}_j \right\|^2 \in \mathbf{R}; 1 \leq i < n; i < j \leq n$, where Δx_k is the distance between a pair of values $\mathbf{X}_i$ and $\mathbf{X}_j$. In the same way lets estimate the set Δy containing the distances of all output elements y. Lets now define a delta set as $\Delta D = \{(\Delta x_1, \Delta y_1) \cdots (\Delta x_k, \Delta y_k) \cdots (\Delta x_l, \Delta y_l)\}$, which contains all distances for the elements of the dataset D.

It is to expect that if two values in the input domain are different, then their corresponding values in the output domain should be similarly different as well. Generalizing this idea it is possible to affirm that "the more different the values are in the input domain, the more different the values should be in the output domain as well", consequently, higher values of Δx_k trend to produce higher values of Δy_k. Therefore the relationship between each Δx_k and Δy_k in the delta set ΔD trends to be proportional. But the concept of consistency is not quite accurate yet; so far this definition does not contemplate the case of having $\mathbf{X}_i = \mathbf{X}_j; y_i \neq y_j$, meaning that $|\Delta x_k| > 0; |\Delta y_k| = 0$, such case should be perfectly valid in the real world domain, however the situation where $\mathbf{X}_i = \mathbf{X}_j; y_i \neq y_j$ should never occur, otherwise it could not be possible to describe the dataset D as Equation 28. Lets then refine the definition of consistency by adding that $(|\Delta x_k| = 0)$ implies that $(|\Delta y_k| = 0)$ and that $(|\Delta x_k| > 0)$ implies that $(0 \leq |\Delta y_k|)$. In other words, "equal observations in the input domain should not have different associated values in the output domain", and in the same way, "different observations in the input domain should have equal or different associated

values in the output domain". It is possible then to define inconsistency in a given data set if it cannot be described as Equation 28 considering the assumptions previously stated.

Sometimes it is not possible to derive a model in the form of Equation 28 to describe a dataset obtained from the real world either because the dataset is incomplete or because there is not enough information available. In the first case, it would be required to obtain more observations of the phenomena. In the second case, a suitable method to make the data consistent would be necessary. At first sight it may look like the dataset is inconsistent, however there is the possibility that some information is missing or it is not available. The method described in this chapter assumes that some information is not available and therefore absent in the dataset (second case), the task then is to find such information using the data that is currently available.

The following section describes the methodology to find a consistent dataset and a suitable function for it, and finally it describes a variation of the methodology to cluster inconsistent datasets. Later this chapter provides a brief background on Genetic Algorithms (GA).

8.1.1 Methodology to Find Consistent Data sets and Their Models.

Let us then define the existing dataset as $S = \{(\mathbf{X}_1, y_1), \cdots (\mathbf{X}_i, y_i) \cdots (\mathbf{X}_n, y_n)\}$, where each $\mathbf{X}_i \in \theta \subset \mathbf{R}^M$, being M number of dimensions "attributes" the input $\mathbf{X}_i$ should have to make S consistent, and θ the suitable values of X_{iM}. Since there is some missing information, we assume that each $\mathbf{X}_i$ is incomplete and that at least one of its values "attributes" is missing.

It is also necessary to define an appropriate model in accordance to Equation 28 to find the missing information of the dataset S, lets define such model as:

$$y'_i = f(\mathbf{X}_i; \boldsymbol{\beta}) + \varepsilon_i \forall i; 1 \le i \le n$$

Equation 29: Proposed model example

This method takes advantage of the Genetic Algorithm (GA) to find such missing information $X_{iM} \in \theta \subset \mathbf{R}$ for each observation $\mathbf{X}_i$. Let us consider G as the set of possible solutions for Equation 29 taking into account dataset S; in other words G is the population (chromosomes) of the GA, so we can define $G_v = \{X_{1M}, \cdots X_{iM} \cdots X_{nM}\}; 1 \le v \le V$, where V denotes the population size. After each generation the GA proposes a new set G as potential solutions for Equation 29, then each Gv is merged to S to form the complete set of observations. It is also required to find the appropriate parameters $\boldsymbol{\beta}_m$ that suit the model Equation 28 correctly; the way to find all parameters $\boldsymbol{\beta}_m$ will depend on the particular model chosen. Each chromosome Gv is then evaluated as part of Equation 28 to observe its behaviour. The fitness function of the GA should minimize the error between the output data y in set S and the result of the model being constructed (2), therefore,

$$fitness_function = \sum_{i=1}^{n} (y_i - y'_i)^2$$

Equation 30: Fitness Function

The GA provides then the missing information vector $\mathbf{X}_{iM} \in \theta \subset \mathbf{R} \forall \mathbf{X}_i; 1 \le i \le n$ and the model $y'_i = f(\mathbf{X}_i; \boldsymbol{\beta}) + \varepsilon_i$ that best fits the data in the consistency sense; each value X_{iM} corresponds to each observation in the original dataset. The method can be described with the following general steps:

1. Extract the input values of data set *S*.
2. Merge the best solution (extra variable) supplied by GA to the extracted input values of set *S*.
3. Obtain the model by using an appropriate regression method.
4. Compute *y'* from the new model.
5. Feedback the GA with *y'* to compute the fitness function, Equation 29, for each element in the population.
6. If the best fitness function does not meet the expectations go to step 1, otherwise end.

A set of values X_{iM} is added to the input data set in *S* to obtain a regression model in the form of Equation 28 with appropriate parameters $\boldsymbol{\beta}$. Such regression model is then used by the GA to compare it with the output *y* of dataset *S*. The GA provides then a new set of solutions X_{iM} that best fits the model, Equation 28, and that reduces the error specified in the fitness function in Equation 29. At the end, the GA provides the missing information $X_{iM} \in \theta \subset \mathbf{R}$ and the best model $y'_i = f(\mathbf{X}_i; \boldsymbol{\beta}) + \varepsilon_i$ found for the inconsistent dataset *S*.

8.1.2 Methodology to Cluster Inconsistent Data sets.

Besides finding a suitable model and the missing information for the new consistent dataset, this method can be used as a clustering algorithm. Let *c* be defined by the user and represent the number of clusters to find, and E_c the set of elements $\mathbf{X}_i$ for cluster *c*, then if the number of permissible values θ for X_{iM} are limited to *c*, and θ is chosen from the universe of $\mathbf{R}$; then the GA provides $X_{iM} \in \theta \subset \mathbf{R}$, and the clusters $E_c = \{\mathbf{X}_i | X_{iM} \in \theta_c\}$. Lets also note that if the Universe of $\theta \subset \mathbf{R}$, then *c* trends to assume *n* different values, therefore $\|y - y'\|^2$ trends to be zero. In this way the GA splits

the elements of the dataset S into subcategories E_c, where each E_c corresponds to a particular hyperplane (label) c, and contains similar elements that require the same $X_{ai} \in \theta_c; 1 \le i \le q$, where q is the number of elements of set E_c, to make the data $\mathbf{X}$ consistent to y. However it is possible that $\|y - y'\|^2 \neq 0$ for clustering purposes, because θ may not include the most adequate values for the particular dataset S according to the specified model in Equation 28. At the end, this method provides the sub sets $\mathbf{X} \in E_c \forall c$ and the best model, Equation 28, that best fits the dataset S.

8.2 Background on Genetic Algorithms

A Genetic Algorithm is a high level simulation of a biologically motivated adaptive system, namely evolution [9]. The algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions are chosen according to their fitness, the more suitable they are the more chances they have to reproduce and survive. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied [28].

Essentially, Genetic Algorithms (GAs) are methods of computer programs and solutions with the intention to optimize or search solutions of a certain problem by means of simulated evolution. Processes loosely based on natural selection, crossover, and mutation are repeatedly applied to a population of strings which represent potential solutions. Over time, the number of above-average "stronger" individuals increases, and better-fit individuals are created, until a good solution to the problem at hand is found [29]. The Genetic Algorithms are often viewed as function optimizers, although the range of problems to which this technique has been applied is quite broad.

An implementation of a Genetic Algorithm starts with the creation of a population (chromosomes) with or without fixed size; the population is usually generated randomly. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes, which represent a better solution to the target problem, are given more chances to "reproduce" and "survive" than those chromosomes that are poorer "weaker" solutions. The "goodness" of a solution is typically defined with respect to the current population [6].

There are basically two challenges a researcher will face when using genetic algorithms, the encoding problem and the evaluation problem. The encoding consists on representing the variables under study using a finite code, usually bit codes. As for the evaluation problem, the key is to be able to evaluate the goodness "strength" of the solutions in a given population; sometimes this is not an easy task. The evaluation function must be capable of measuring adequately the ability of a solution to solve the problem. If the evaluation function fails the entire algorithm most likely will fail.

There are three basic operators in the genetic algorithms that are executed sequentially [5]:

- Selection and Reproduction
- Crossover
- Mutation

For the selection and reproduction, pairs of individuals are selected from the population according to their "strength". This operator can be implemented in a variety of ways; one of the common methods is the roulette wheel selection. This type of selection simulates a spinning wheel; each portion of the will is assigned to each individual proportionally to its "strength" so the better individuals have better chances to pass to the next generation. The number of individuals to pass to the next generation when using this method is fixed by the user. The selection can be also complemented with "elitism". The "elitism" consists on copying the strongest or many stronger individuals to the next generation without going trough the spinning wheel; this guaranties an increasing evolution throughout the overall process [5].

Crossover and mutation have a fixed rate of occurrence, so the operators are applied with a certain fixed probability. The researcher must specify these probabilities according to the specific problem. Having a high probability of crossover leads to a more diverse population but not necessarily a better one. Specifying a high probability of mutation could cause that many of the new elements in the population do not reflect a direct evolution from the parents [5].

The crossover is implemented after the selection and reproduction. During the crossover the randomly selected chromosomes "parents" are combined to produce new chromosomes "children"; which hopefully have a higher fitness than their parents. The parent's chromosomes are divided into two strings randomly, then the new strings are combined and the children are generated. The parent's chromosomes must be combined in such a way that each pair of parents will produce only two different children, so the overall population does not grow or diminish [5].

The mutation operator is applied to every single string coming from the crossover process. During the mutation process, each character of the chromosome has a very low probability of changing to a different random value of the same kind and range. The mutation is a very useful process to avoid missing high fitness stings when the current population has converged to a local optimum, so mutation spreads out the diversity of the population and explores new regions in the search space [5].

The following are the general steps implemented when using genetic algorithms:

1. Generate randomly an initial population

2. Generate a new population by applying the selection and reproduction operators to select pairs of strings. The number of pairs will be the population size divided by two, so the population size will remain constant between generations.

3. Apply the crossover operator to the pairs of the individuals of the new population.

4. Apply the mutation operator to each individual in the new population.

5. Replace the old population with the new population.

6. Copy the best-fitted individual(s) to the newly created population to warrantee evolution.

7. (If the number of iterations is less than the maximum go to step two, else stop) OR (If the fitness of the best result does not get better over certain number of iteration, then stop) [5].

8.3 Experimentation and Results

This method is first tested using artificial datasets to validate its effectiveness and behaviour, then the method is used to obtain the missing information and the models for the software data sets discussed in this thesis.

The method is first used to find from two to ten different clusters in an inconsistent artificial dataset, then it is used to find a solution that makes the data consistent without performing any clustering. The method is then tested in finding suitable models and the missing information of the software datasets obtained from the industry to validate its performance. Finally the results are compared and discussed.

8.3.1 Experimentation With Artificial Data

Let S be an inconsistent artificial dataset represented as $S = \{(X_{1m}, y_1), \cdots (X_{im}, y_i) \cdots (X_{nm}, y_n)\}$, where each $\mathbf{X}_m \in \boldsymbol{R}; 1 \le m \le M-1$. Let y' be the model to represent the data:

$$y'_i = f(\mathbf{X}_i; \boldsymbol{\beta}) + \varepsilon_i = \beta_0 + \sum_{m=1}^{M} \beta_m X_{im}$$

Equation 31: Proposed model

Then the task is to find the values of all $\mathbf{X}_{iM} \in \theta$ such that the dataset S can be expressed in the form of Equation 31. To do so, the method will use the Genetic Algorithm to find the values of X_{iM}, and the Least Squared Error (LSE) method to find the values of $\boldsymbol{\beta}$ in

Equation 31. To create the artificial inconsistent dataset *S*, we first generate a consistent dataset *D* whose output set *y* is produced by the following function:

$$y_i = 0.8X_{i1} + 0.2X_{i2} + 0.1X_{i3} + X_{i4}$$

Equation 32: Artificial data set

Where $\forall i \in [1,\cdots,256] : \mathbf{X}_i \in [0,1] \subset \mathbf{R}^M$, and are random numbers. Then the components of the first term ($0.8X_{i1}$) are taken away from all elements in the input domain, making the new data set *S* inconsistent by itself according to the explanation stated above. After calculating the delta sets for the original data set *D* and the inconsistent dataset *S* the inconsistency becomes evident. Figure 60 and Figure 61 show the delta sets for *D* and *S*.

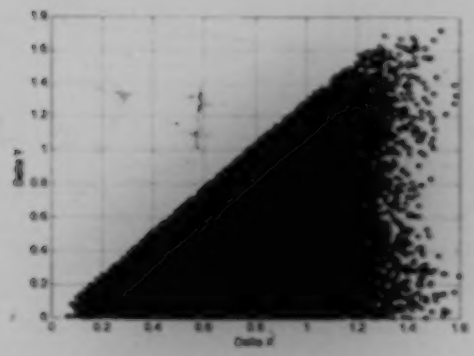


Figure 60: Delta set for *D* (consistent)

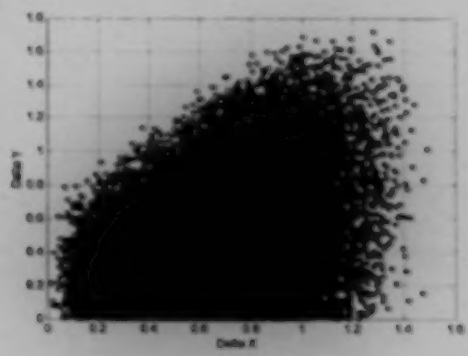


Figure 61: Delta set for *S* (inconsistent)

The delta set for *D* (Figure 60) presents the distances between all input elements vs the distances between all output values, in this plot all delta elements are in the lower right side of the plane, indicating that for similar values in the input domain there are corresponding similar values in the output domain. The delta set for *S* (Figure 61) presents the case of an inconsistent data set obtained by taking one dimension away from the consistent data set *D*, there the relationship between inputs and outputs is not as clearly defined as in that presented for delta set *D* (Figure 60), the delta set for *S* (Figure 61) shows that there are not similar output values for similar input values (upper left part of the plane), therefore the data set *S* is inconsistent according to the definition previously specified.

Notice also that the lower right area of the delta plots presents not only that similar input elements have similar output elements, but also that there could be similar output values for different input observations, this is perfectly valid as specified by the consistency definition previously stated.

Lets now define the GA appropriately for each set of clusters to find, and according to the following characteristics: Fitness function, Equation 30, to be minimized, probability of

crossover of 0.15, probability of mutation of 0.75, a stop criterion when reaching 10,000 generations, when the fitness function = 0, or when the fitness function does not change for 1500 generations, and $X_{iM} = X_{i4} \in \theta \forall \mathbf{X}_i; 1 \le i \le 256$. These characteristics of the GA are not fixed, in fact it is a good practice to experiment with different values for each cluster label, different set of values may produce a more consistent dataset at the end. It is suggested to make several attempts to obtain suitable values for a particular dataset.

Table 20 shows the characteristics for the method considering the number of clusters to obtain $E_c = \{\mathbf{X}_i | X_{id} \in \theta_c \subset [0 \cdots 1]\}$.

c clusters	θ
2	[0, 1]
3	[0, ½, 1]
4	[0, 1/3, 2/3, 1]
5	[0, 1/4, 2/4, 3/4, 1]
6	[0, 1/5, 2/5, 3/5, 4/5, 1]
7	[0, 1/6, 2/6, 3/6, 4/6, 5/6, 1]
8	[0, 1/7, 2/7, 3/7, 4/7, 5/7, 6/7, 1]
9	[0, 1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8, 1]
10	[0, 1/9, 2/9, 3/9, 4/9, 5/9, 6/9, 7/9, 8/9, 1]

Table 20: Characteristics for X_{id}

8.3.2 Results for the experimentation with artificial data

For the clustering algorithm, the fitness function is successfully minimized as the number of clusters is increased, when θ is allowed to assume any limited **R** the fitness function minimizes the most, as expected; the values for the fitness function depending on the number of clusters are presented in Figure 62, where the values decrease as the number of

clusters increases, since the granularity of the chromosome is refined the error is then reduced.

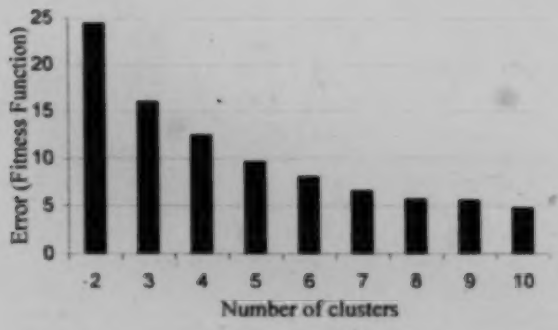


Figure 62: Performance of the clustering algorithm with artificial data

Since the size of the data sets has an impact on the measures of errors for the clusters, the values of the error can be divided by the number of observations in the data set. This makes possible to compare data sets of different sizes in an objective way, however the errors are not divided for the results of the artificial data sets since it is not to be compared to any other data set, it is just to validate the performance of the method. Later in this chapter, when the method is used to experiment with data from the industry, the errors "squared differences" are to be divided by the number of element in each data set to compare the results from project to project.

Table 21 presents the parameters of the models for different number of clusters; the values of the coefficients β do not change dramatically since the only difference in the model is the number of clusters.

c clusters	Model $y_i' = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \beta_4 X_{i4}$				
	β_0	β_1	β_2	β_3	β_4
2	0.210	0.228	0.067	0.996	0.404
3	0.676	0.207	0.0	1.040	– 0.537
4	0.688	0.281	0.023	1.050	– 0.605
5	0.755	0.213	– 0.009	0.962	– 0.614
6	0.809	0.103	0.070	1.020	– 0.706
7	0.002	0.270	0.066	0.984	0.695
8	0.089	0.189	0.052	0.957	0.722
9	0.760	0.232	0.042	0.949	– 0.701
10	0.727	0.170	0.080	0.966	0.716

Table 21: Models and fitness function

By plotting the resulting y' vs y it is possible to appreciate the similarities between the original values of X_{iM} and the values obtained by the GA. If the values of y' and y are complete equal then a 45 degree line is depicted, otherwise the elements are scattered throughout the plot in proportion to their differences. Figure 63 to Figure 66 present the obtained plots of y' vs y when this method is used as a clustering technique, it shows the plots for only 2, 5, and 10 clusters as well as the plot of y' vs y when the method is used to find X_{iM} among any limited **R** value such that S becomes consistent.

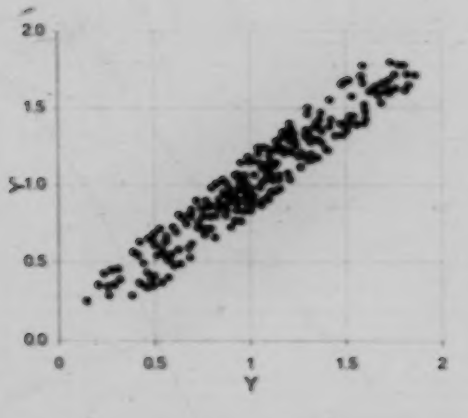


Figure 63: Accuracy for 2 clusters

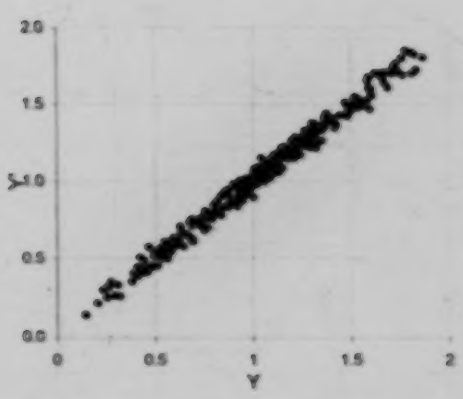


Figure 64: Accuracy for 5 clusters

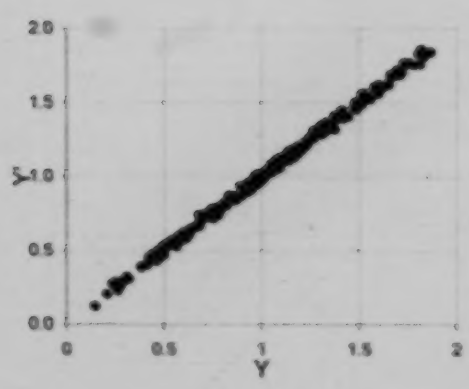


Figure 65: Accuracy for 10 clusters

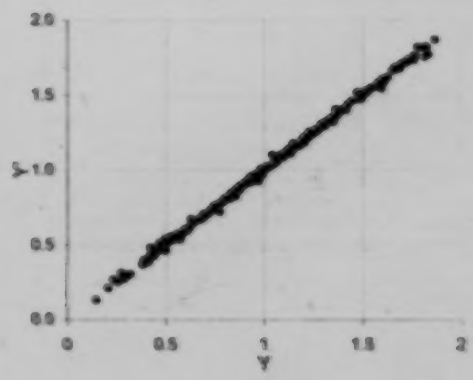


Figure 66: Accuracy for n clusters

Figure 62 shows the tendency of minimizing the fitness function as the number of clusters is increased, when the method is not used to cluster data but to find missing information in the dataset, the fitness function reaches its minimum value. Figure 63 to Figure 66 show the tendency to increase the similarity between y' and y as the number of clusters is increases, to finally reach a best sólution for the inconsistent dataset when no clustering is considered. This is expected if the values of θ are such that allow X_{iM} to assume values from a larger variety. When $\theta \in \mathbf{R}$ then it is possible to find missing information for the dataset, this can also be seen as if the clustering method is allowed to label the data within infinite possible classes, therefore each element from S is categorized with a unique label, making the data consistent.

8.3.3 Experimentation With Data From the Industry

This section describes the experimentation and results of this method using the software data sets for this thesis.

Lets now refer to each software data set as $S = \{(\mathbf{X}_1, y_1), \cdots (\mathbf{X}_i, y_i) \cdots (\mathbf{X}_n, y_n)\}$, where $\mathbf{X}_i \in \mathbf{R}^6$ denotes the six CK-metrics previously described and y_i the number of modifications made to the software classes. Let now the model to describe each data set as:

$$y'_+ = f(\mathbf{X}_i; \boldsymbol{\beta}) + \varepsilon_i = \beta_0 + \sum_{m=1}^{7} \beta_m X_{im}; 1 \le i \le n$$

Equation 33: Proposed model for software data

As before, the task now is to find suitable values for $\mathbf{X}_{i7} \in \theta \in \mathbf{R}$ such that the dataset S can be expressed in the form of Equation 33. Lets now define the parameters for the GA in the following way: Fitness function, Equation 30, to be minimized, probability of crossover of 0.15, probability of mutation of 0.75, a stop criterion when reaching 10,000 generations, when the fitness function = 0, or when the fitness function does not change for 1500 generations, and $X_{iM} = X_{i7} \in \theta \forall \mathbf{X}_i; 1 \le i \le n$.

Table 22 shows the feasible values of θ for different number of clusters to obtain $E_c = \{\mathbf{X}_i | X_{i7} \in \theta_c \subset [0 \cdots 1]\}$.

c clusters	θ
2	[0, 1]
5	[0, 2/4, 2/4, 3/4, 1]
10	[0, 1/9, 2/9, 3/9, 4/9, 5/9, 6/9, 7/9, 8/9, 1]

Table 22: Characteristics of X_{iM}

8.3.4 Results for the Experimentation With Data From the Industry

In all cases the error decreases as the number of clusters increases, Figure 67 depicts the errors "fitness function" for all software projects depending on the number of clusters, 2, 5, 10, and *n*. In general this method is most successful with project E, the error for *n* clusters is the closest to zero, however this project is the smallest one from all.

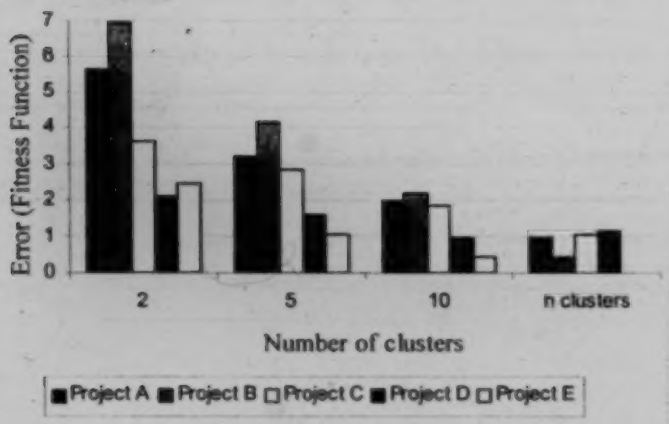


Figure 67: Errors of software projects according to number of clusters

Table 23 and Figure 68 present the average of the squared differences between the model's output and the actual number of fixes, according to Equation 16. This is, the fitness function divided by the number of elements in each data set, doing this allows us to compare fairly the models regardless the number of observations in the data sets.

c clusters	Project A	Project B	Project C	Project D	Project E
2	0.0609	0.0579	0.0357	0.0469	0.0646
5	0.0346	0.0349	0.0277	0.0358	0.0276
10	0.0211	0.0181	0.0181	0.0213	0.0113
n	0.0106	0.0036	0.0102	0.0255	0.0003

Table 23: Averaged squared differences

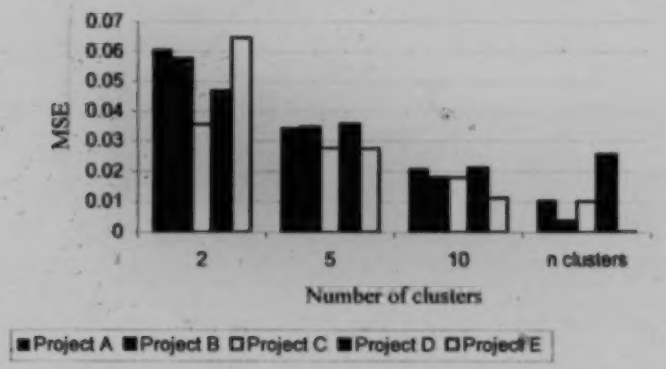


Figure 68: Averaged squared differences or MSE

The models proposed for the software data sets are of the form *fixes* = $\beta_0 + \beta_1 NoM_i + \beta_2 DIT_i + \beta_3 NOC_i + \beta_4 CBO_i + \beta_5 RFC_i + \beta_6 LCOM_i + \beta_7 X_{i7}$ as described in Equation 33, Table 24 presents the coefficients of the linear models for each project for n clusters to find the missing attribute. Lets remember that the data is normalized in the rage [0, 1] in the input and output domains independently, therefore this coefficients provide a solution in such range. The introduced variable X_{i7} has similar impact as some of the rest of the attributes, specially the offset β_0, and β_6. The coefficients β_7 are small with respect to the rest of the coefficients so this attribute is not taking over the rest of the elements in the model, however the coefficients β_2 and β_3 are, in general, high compared to the rest.

Coefficients	*Project*				
	A	**B**	**C**	**D**	**E**
β_0	0.0350	-0.3940	-0.1804	0.5413	0.6751
β_1	-18.6758	-16.5361	-124.9871	1.4329	-53.7746
β_2	-4.4828	242.2207	-1349.3465	-677.3945	-1566.5869
β_3	2.7807	-265.2064	-108.3969	71.8986	692.9627
β_4	-3.1337	-39.5710	51.0822	-14.3795	6.4388
β_5	3.4626	13.6675	60.7714	8.3000	7.8088
β_6	1.2744	0.2102	0.3898	-0.4778	1.8236
β_7	0.6396	1.3642	0.5378	-0.6411	-0.8048

Table 24: Coefficients of the linear models

To better appreciate the accuracy of the models it is convenient to plot the results of the models vs. the actual number of modification provided in the data sets; if the models are completely accurate then the plots should depict a 45 degree line in the range [0, 1] since the data is normalized. Figure 69 to Figure 73 present the plots of the accuracy for the software projects.

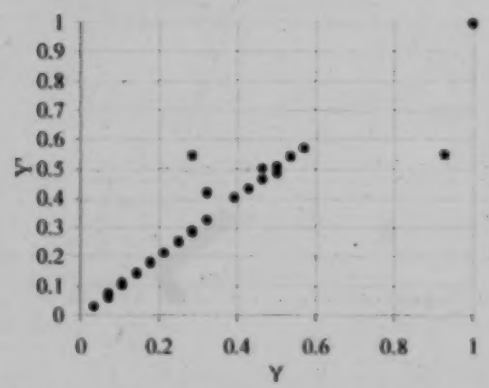


Figure 69: Consistency for project A

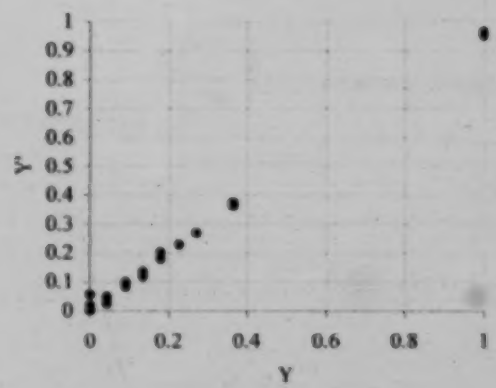


Figure 70: Consistency for project B

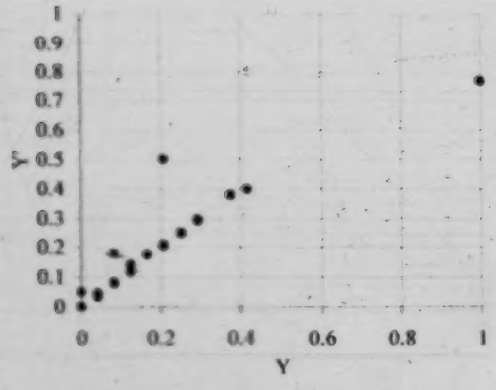


Figure 71: Consistency for project C

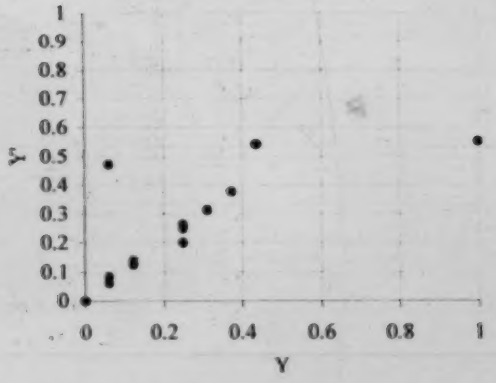


Figure 72: Consistency for project D

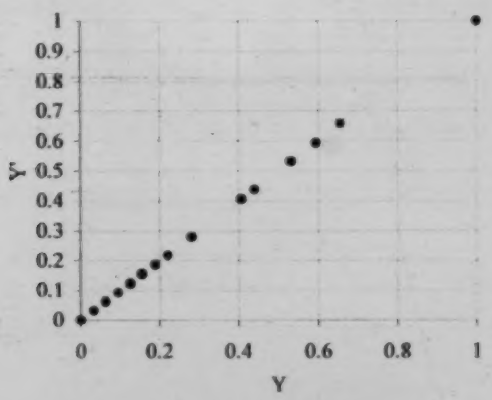


Figure 73: Consistency for project E

It is to expect not to have a completely accurate linear model for each software project because the data sets are very inconsistent, as described in "Morphology of the data", however they should depict values close to the actual number of modifications. In order to better understand the behaviour of the method the solutions are proposed as linear models in this thesis, nevertheless it is possible to obtain better models to fit these datasets if they are of different nature. In this particular case the models are accurate in most of the situations, as depicted in Figure 69 to Figure 73, however there are some elements beyond the 45-degree line that make the models inaccurate for few input values.

8.4 Conclusions

This chapter presents a method that proves to be powerful for finding missing information in inconsistent datasets as well as mathematical models to describe the new consistent records; and with a small variation in the parameter θ the method can be used as a

clustering algorithm as well. The values of θ have to be carefully chosen so that the new information makes the model accurate enough for specific purposes, they also have to reflect some attributes that are missing in a realistic way, this is, they have to reflect a characteristic of the phenomenon of interest that cannot be measured or that is not available, not only arbitrary numbers.

This method has successfully found the missing attribute in the software data sets according to the lineal model proposed in Equation 33, the new attribute X_{i7} is not taking over the rest of the variables in the data sets, as reflected in the values of its coefficient β_7. Nevertheless there are some elements in the accuracy plots depicted in Figure 69 to Figure 73 that are far from the expected 45-degree line, however they are only very few compared to the amount of total points in the data sets. Lets also notice that models of different nature can be specified as solutions for the software data, however linear models are used in this research to better understand and comprehend the behaviour of this method.

It is important as well that the values of each X_{iM} do not take over the rest of the attributes in the observations for a specific new model, one way to overcome such problem is to normalize the complete set and to let the values of θ be in the range of the minimum and maximum values of the normalized data set. Another approach can be to let θ assume values between the minimum and maximum values of the raw data set.

We believe that this method is not restricted to the software engineering filed but it can be applied to any type of data. In the particular case of software engineering, it could be very helpful to find relevant missing information that leads to produce better models. The new data can be used as a baseline to find the missing information in the real phenomena by purely knowing its characteristics, or it can be used as part of the information that cannot be obtained in experimental way because it is simply not available.

When the researcher knows what kind of information is missing then the task of finding it from the phenomenon's source becomes easier, the information provided by this method can serve as a guide to know the characteristics of the data that is not present in the records, and can give a hint on where to look for it. It is important to note that the new data is found based in the proposed definition of consistency, but the method still works if a new definition is introduced. The accuracy of the model that this method provides depends in great part in the characteristics chosen for it, a good practice is to try different models to search for those that suit the best a particular phenomenon.

Discussion

The objective of the thesis is to investigate applicability of computational intelligence based techniques to analysis of industrial software engineering data. This work presents a number of approaches to model software engineering data in order to predict the number of software modifications. The work focuses on finding a model or a set of models able to characterize the software engineering data. Several models and algorithms are examined. The data comes in the form of software CK-metrics, representing the design of objects in C++ programming language from particular projects in the telecommunications domain.

The following techniques are used to analyze software data and develop software models:

1. Clustering Using the Fuzzy *c* Means Algorithm
2. Multivariable Regression
3. Clustering and Local Regression
4. Feedforward Neural Networks Using the Backpropagation Learning Rule
5. Switching Regression Models and Fuzzy Clustering
6. Genetic Algorithm-Based Clustering Method

The objective of the fuzzy clustering is to split the data into two basic groups, one containing the observations with zeros associated in the output domain, and another one enclosing the rest of the data. In this way, the clustering would provide two different subsets of information that could be model separately. Since the data is zero inflated, this would provide an advantage in terms of accuracy to the models.

Unfortunately the fuzzy c means algorithm is not able to split the data sets into the desired groups, which leads to the option of finding models for the complete data sets. The observations in the data are hardly distinguishable; the observations containing zeros are not easily differentiable from the observations with non-zero data. The clustering step is used as a pre-processing of the data and is not required prior the rest of the approaches.

Some of the methods described in this work are different in nature, and they cannot be compared directly to each other, for such reason, the performance of the Multivariable regression, Neural Networks, and Genetic Algorithm-Based Clustering methods are described in the same way, they all use the average of summed squared errors as a performance index, hence they can be compared to one another. In the same way, the Clustering and Local Regression, Regression Models and Fuzzy Clustering, and the

Genetic Algorithm-Based Clustering methods are described in terms of error, but the error is computed for each cluster independently, so they can be compared.

Figure 74 presents a comparison chart of the performance, in terms of error, of the Neural Networks (including the five approaches), the Multivariable regression, and the Genetic Algorithm-Based Clustering Method.

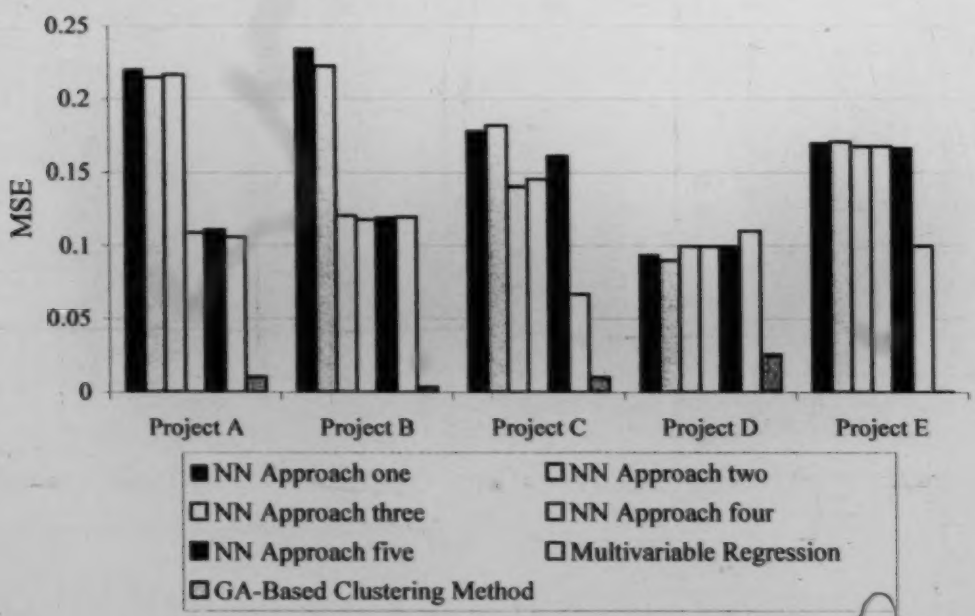


Figure 74: Comparative chart for NN, Multivariable Regression, and GA-Based Clustering Method

It is very interesting that the error is the highest when modeling with neural networks, one would expect to have better results using this technique in the data sets. However, lets remember that the data sets are zero inflated, inconsistent and small. This is a reason that the neural networks have difficulties to learn from the observations. The neural networks require large data sets to be able to learn the essence of the data, and since the software data used in this work are small, the neural networks are not able to build successful models.

Even tough the neural networks are not doing a good job in the prediction of the defects in the software data sets, there is a significant improvement in their performance when the constant weight modifier in the backpropagation learning equations is used. This suggests that the zero inflation of the data sets are playing a very important role in the modeling of the data. When the constant weight modifier is introduced to the learning rule the neural networks improve their accuracy, this makes clear that the zero inflation phenomenon can be controlled, up to some extent, by this technique.

As for the Multivariable Regression, the errors of these models are comparable to those of the neural networks, this situation may not be of surprise if we take into account the zero inflation of the data sets. These results suggest that the linear models are mainly mapping the observations that contain zeroes in the output domain. The magnitudes of the coefficients of the models suggest that the attributes that are highly correlated to the number of fixes are mostly responsible for the prediction of the fixes. In a way, some attributes are overtaking the rest of the input variables. It is important to mention that this approach provides a comprehensible model of the software data sets, which is an advantage over the neural networks approaches.

It is possible to see in Figure 74 that the lowest error belongs to the Genetic Based-Clustering Method. This method introduces a new attribute to the data set such that the data becomes consistent. This is a new and an important characteristic of this method with respect to the rest. By introducing the extra attribute, the data set does not longer have observations with same inputs and different outputs. Therefore, the complete data set can be used to build the software models. The results suggest that the software data has missing information. There may be variables that are not taken into account when measuring the software. It is possible that these variables are not present at the moment of the design of the software or are not obvious. For instance, one can suggest that time pressure to finish the software products makes an impact in the programmers, or that changes in the design during the development or testing phases introduced some errors in the code, situation that may explain up to some degree the inconsistency of the data sets. Another variable that has not been considered is the experience and motivation of the programmers. In this regard, we do not have the information as if the teams of programmers were the same for all projects or any further information as how the data was collected.

By finding the missing variable in the data sets with the Genetic Algorithm-Based Clustering Method, the researcher is able to know the type of information that is missing, for instance, he/she can know the form and distribution of such information, making it easier to find the missing attribute. This is probably the greatest advantage of this method.

As for the rest of the methods, the Clustering and Local Regression provides a set of sub models to represent the whole data set. A linear regression model is found for each cluster of data. This method has the advantage of providing comprehensive mathematical models to describe the observations, but its accuracy depends greatly on the goodness of the clusters.

In this work it was found that the elements are not normally distributed and that they are inconsistent (Chapter 2). So the Clustering and Local Regression does not provide the most accurate solution if it is compared to the rest of the computational intelligence techniques, and does not provide a unique model to describe the data, however it can deal with the fact of having identical inputs with different outputs.

The Clustering and Local Regression Method can be easily compared to the Switching Regression Models and Fuzzy Clustering algorithm, both methods can find models that

provide different solutions for the same CK metrics, these two methods can also provide models for each independent cluster of data, and these two methods can find the models for each dimension separately; however the main difference is that the Clustering and Local Regression first groups the data and then builds a model around the cluster's data; while the Switching Regression Models and Fuzzy Clustering method finds a model and the cluster's data at once. In fact, the clusters' prototypes are a model themselves. In this particular case, this algorithm is applied to the software data using one dimension at a time and their associated number of fixes, obtaining at the end a linear regression model to represent each dimension.

The Switching Regression Models and Fuzzy Clustering is a good tool when dealing with data that contain different outputs for the same inputs. However its biggest limitation is its sensitivity to the morphology of the data and its dependency on the initial position of the prototypes. When dealing with more than 3 clusters the results become almost unpredictable, and it does not find even basic patterns. However it provides comprehensible mathematical models to analyze.

It is hard to say that one of these two methods, "Clustering and Local Regression" or "Switching Regression Models and Fuzzy Clustering", is better than the other, in fact, the error depends on the number of clusters that are formed and, in the case of the Switching Regression Models and Fuzzy Clustering, on the initial position of the models. However it is possible to say that the Clustering and Local Regression method is more predictable and does not have the problem of sensitivity of the initial positions of the prototypes.

Just in the same way it is hard to affirm that one method is better that the other when comparing the Clustering and Local Regression with the Switching Regression Models and Fuzzy Clustering, it is hard to say if the these two methods are better or worst than the Genetic Algorithm-Based Clustering Method, again the error depends on the number of clusters to look for. Nonetheless, one disadvantage of the Genetic Algorithm-Based Clustering Method is that it does not provide models for multiple outputs with the same input. Its advantage is that it does provide a single model that describes the cluster using consistent data sets. This method can actually cluster the data according to the similarity of the information, the clustering depends, in great part, on the values of θ specified by the user, which may lead to unsatisfying results. Prior knowledge of the data may be required to choose valid values of θ.

Overall and based on the results previously presented, it can be said that the software data has tendency to be incomplete, which prevents building accurate models. The recommendation is to look for a variable that has the same morphology of the extra variables introduced by the Genetic Algorithm-Based Clustering Method and to incorporate it to the data sets.

Bibliography and WWW links

BIBLIOGRAPHY

1. Basili V. R., Briand L. C., Melo W. L., "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, 22(10), 1996.

2. Christopher R. Houck, Jeffery A. Joines, and Michael G. Kay , "A Genetic Algorithm for Function Optimization: A Matlab implementation", ACM Transactions on Mathmatical Software, Submitted, 1996.

3. Carl G. Looney, "Pattern Recognition Using Neural Networks, Theory and Algorithms for Engineers and Scientists", Oxford University Press, Inc. 1997.

4. Carolyn Mair, Gada Martin Lefley, *et all,* "An investigation of machine learning based prediction systems", The journal of systems and software, 53, 2000, pp. 23-29.

5. D. Doval, S. Mancoridis, and B. S. Mitchell, "Automatic Clustering of Software Systems using Genetic Algorithm", IEEE Proceedings of the 1999 Int. Conf. on Software Tools and Engineering Practice (STEP'99).

6. Darrell Whitley, "A Genetic Algorithm Tutorial", *Statistics and Computing* Volume 4, pp. 65-85, 1994.

7. Douglas C. Montgomery and George C. Runger, "Applied Statistics and Probability for Engineers", Second Edition, John Wiley & Sons, Inc. 1999.

8. El-Emam, K., Object-Oriented Metrics: A Review of Theory and Practice, NRC/ERB-1085, March 2001. 30 pages. NRC 44190.

9. Fenton N.E., M. Neil, "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, September/October, pp. 675-689, 1999.

10. Fenton N. E. and Pfleeger S. L., "Software Metrics: A Rigorous and Practical Approach", Brooks/Cole Pub Co.

11. Jonathan I. Maletic, Andrian Marcus, "Supporting Program Comprehension Using Semantic and Structural Information", Conference Proceedings, 23rd. ICSE 2001 International Conference on Software Engineering, Toronto, Canada. May 2001. pp. 103-112.

12. Khoshgoftar, Taghi, *et all*, "A comparative study of pattern recognition techniques for quality evaluation of telecommunications in software", IEEE journal. Selec. Areas Comunications, 1994, pp. 279-291.

13. Khoshgoftar, Taghi, Munson J. C., "Predicting Software Development Errors Using Complexity Metrics", IEEE J. Select. Areas Comunications, Vol. 8, (1990), pp. 253-261.

14. Khoshgoftaar, Taghi, *et all*, "Using Neural Networks to Predict Software Faults During Testing", IEEE Transactions on Reliability, Vol. 45, No. 3, September 1996. pp. 456-462.

15. Krishnamoorthy Srinivasan, Douglas Fisher, "Machine Learning Approaches to Edtimating Software Development Effort", IEEE Transactions on Software Engineering, Vol. 21, No. 2, February 1995. pp. 126-136.

16. Krzysztof Cios, Witold Pedrycz, and Roman Swiniarski, "Data Mining Methods for Knowledge Discovery", Kluwer Academic Publishers. 1998.

17. L. Darrell Whitley, "Foundations of GENETIC ALGORITHMS 2", Morgan Kaufmann Publishers, Inc., 1993.

18. Lionel C. Briand, Victor R. Basili, and William M. Thomas, "A Pattern Recognition Approach for Software Engineering Data Analysis", IEEE, Transactions on software engineering, Vol. 18, No. 11, November 1992.

19. Matthew Evett, Pei-der Chien, Taghi M. Khoshgoftaar, Edward B. Allen, "GP-based software quality prediction", *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Koza *et al*, editors, Morgan Kaufmann, 1998.

20. Mauricio A. de Almeida, Hakim Lounis, Walcélio L. Melo, "An Investigation on the Use of Machine Learned Models for Estimating Correction Costs", Forging-New-Links-Proceedings-International-Conference-on-Software-Engineering, IEEE Comp Soc, Los Alamitos, CA, USA. pp. 473-476, 1998.

21. Pekka Abrahamsson, "Commitment Development in Software Process Improvement: Critical Misconceptions", Conference Proceedings, 23rd. ICSE 2001 International Conference on Software Engineering, Toronto, Canada, pp. 71-80, May 2001.

22. Thomas Q. Zeng, Peter Cowell, and Qiming Zhou, "The User's Approach (UshA) for the Realization of Fuzzy Clustering Theory for Zonal Analysis in Raster-Based GIS", Presented at the second annual conference of GeoComputation, 1997 and SIRC 1997, University of Otago, New Zeland.

23. R. J. Hathaway and J.C. Bezdek, "Switching Regression Models and Fuzzy Clustering", IEEE trans. Fuzzy Systems, Vol. 1, No. 3, pp. 195-204, Aug. 1993.

24. Renu Kumar, Suresh Rai, and Jerry L. Trahan, "Neural-Network Techniques for Software-Quality Evaluation", IEEE, Proceedings Annual Reliability and Maintainability Symposium, pp. 155-161, 1998.

25. Robert Hochman, Taghi Khoshgoftaar and John P. Hudepohl, "Evolutionary Neural Networks: A robust Approach to Software Reliability Problems". Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE'97), IEEE September 1997.

26. Succi G., Benedicenti L., Bonamico C., Vernazza T., "The Webmetrics Project – Exploiting 'Software tools on demand'", World multiconference on systemics, Cybemeticsm and Informatics, Orlando, Fl., 1998.

27. W. Hsu and M. F. Tenorio, "Software Engineering Effort Models Using Neural Networks", IEEE 91-IEEE-Int-Jt-Conf-Neural-Networks-IJCNN-91.-Publ-by-IEEE,-IEEE-Service-Center,-Piscataway,-NJ,-USA, pp. 1190-1195, 1991.

WWW LINKS

28. http://cs.felk.cvut.cz/~xobitko/ga/

29. http://www.cs.bgu.ac.il/~omri/NNUGA/

30. http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-12.html

Appendix A

Description and Morphology of the data

Appendix A presents the histograms and cross correlation plots describing the software data sets. The description of the data and its morphology is fully documented in Chapter 2.

Figure A 1 to Figure A 5 show the cross correlation between attributes: LOC, NoM, DIT, NOC, CBO, RFC, LCOM, and Fixes. Table A 1 to Table A 5 depict the histograms each CK-Metric from the input data sets.

Cross correlation of the software data sets

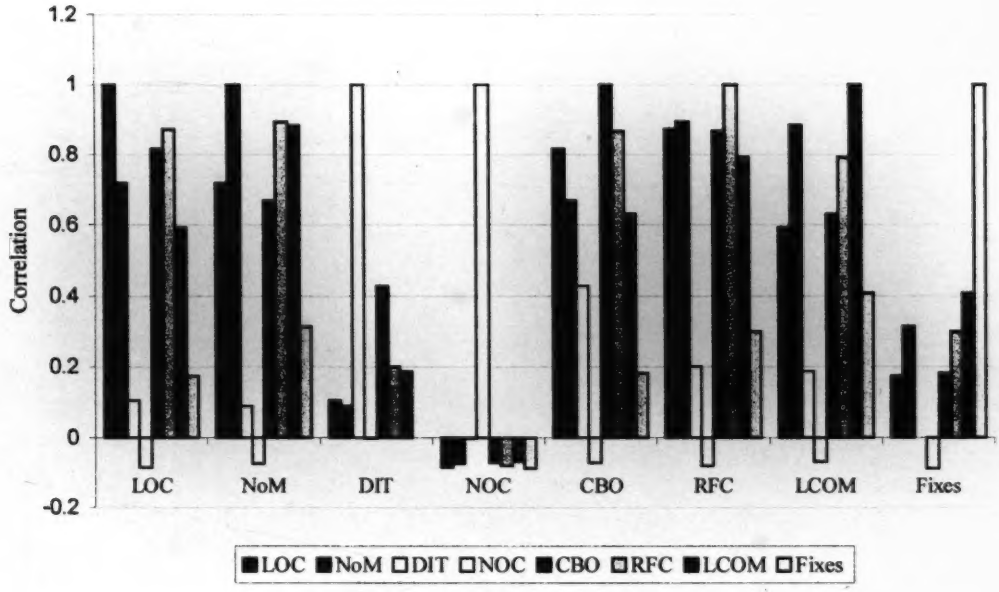


Figure A 1: Cross correlation for project A

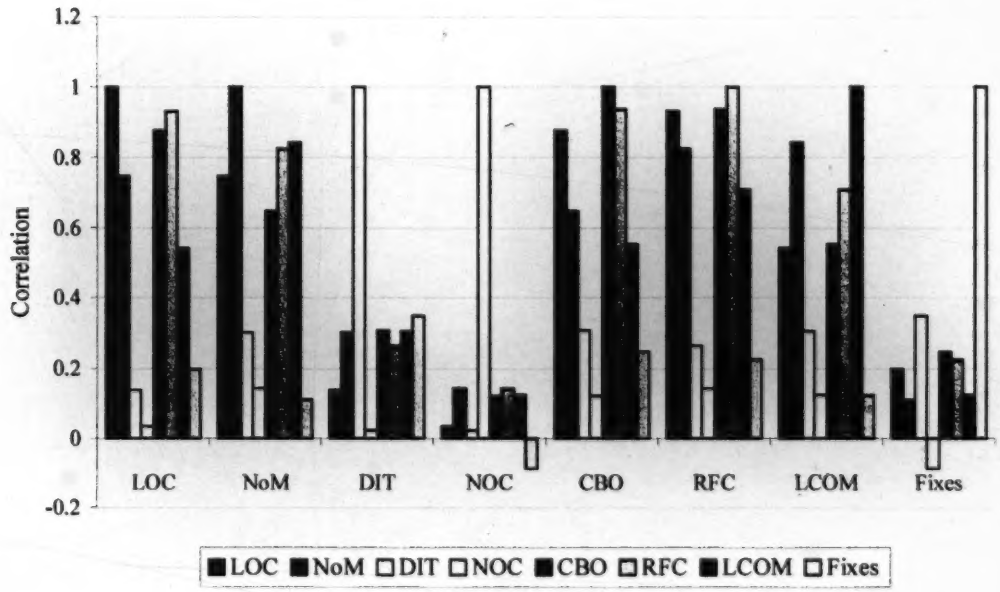


Figure A 2: Cross correlation for project B

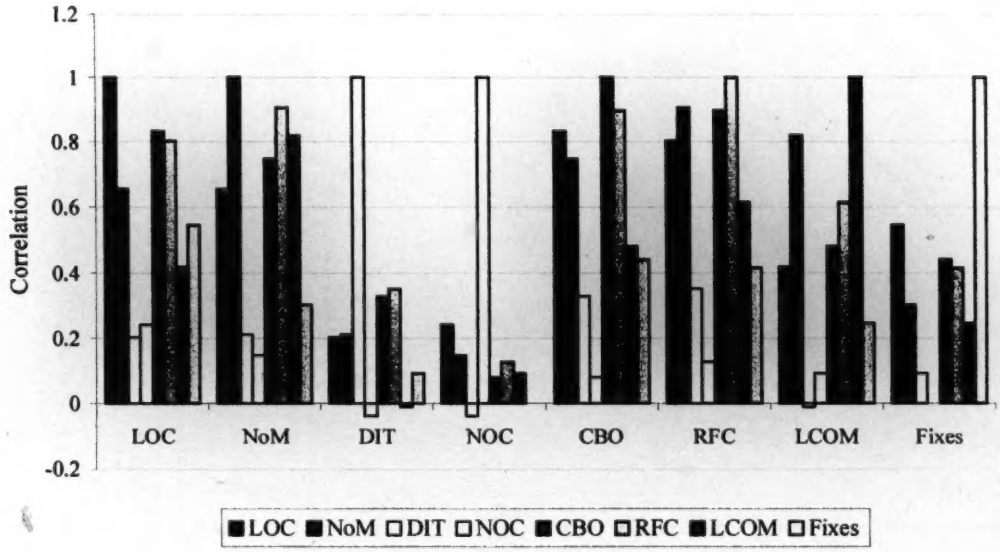


Figure A 3: Cross correlation for project C

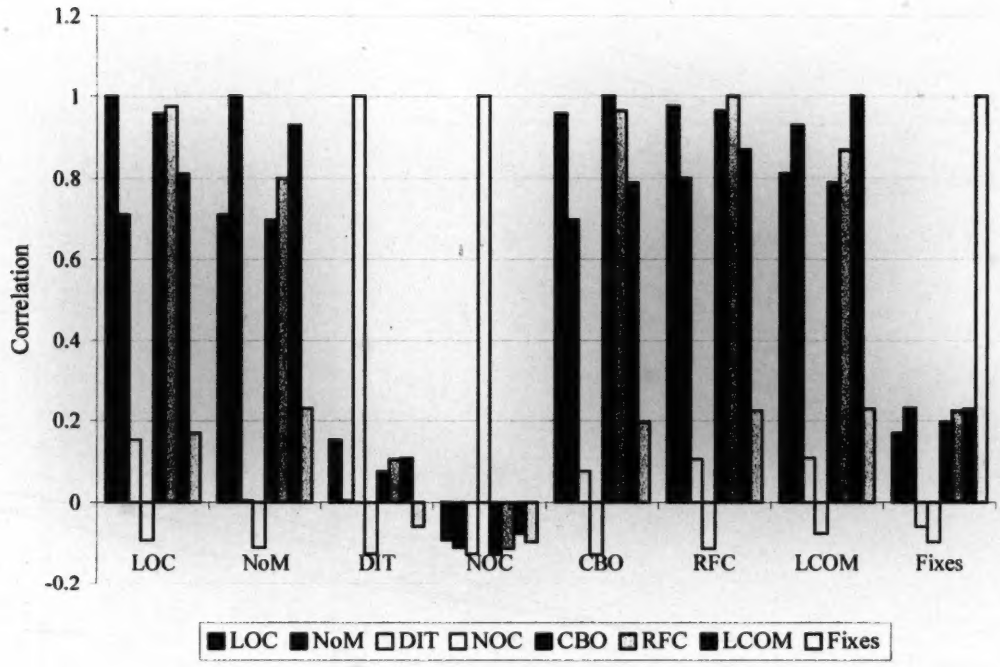


Figure A 4: Cross correlation for project D

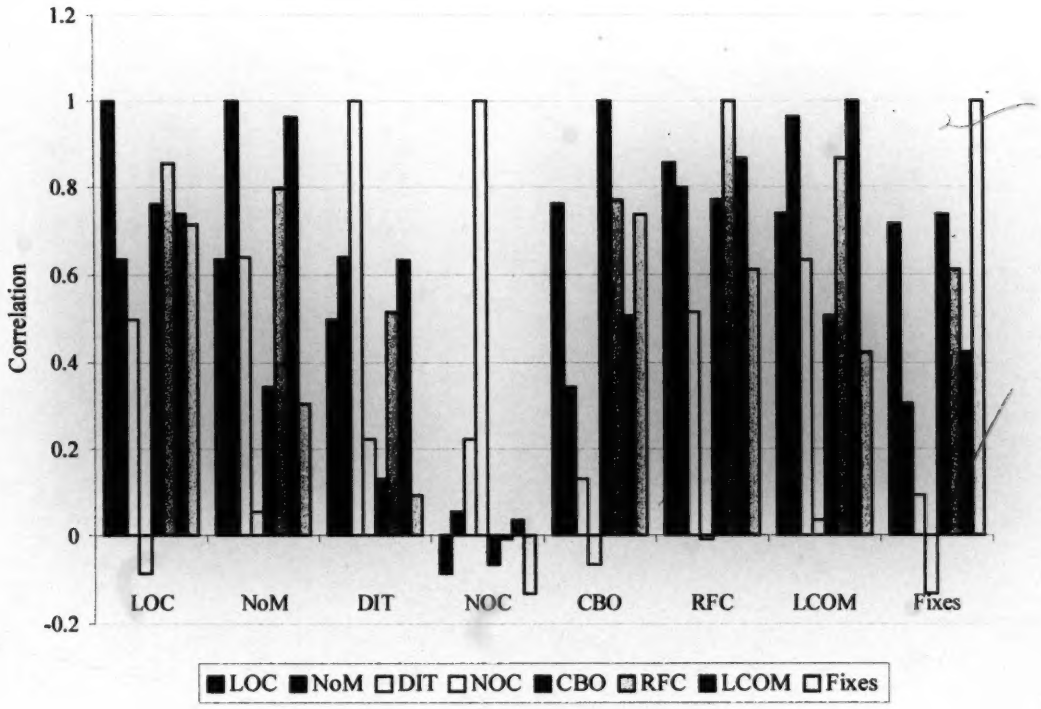


Figure A 5: Cross correlation for project E

Histograms of the software CK-metrics

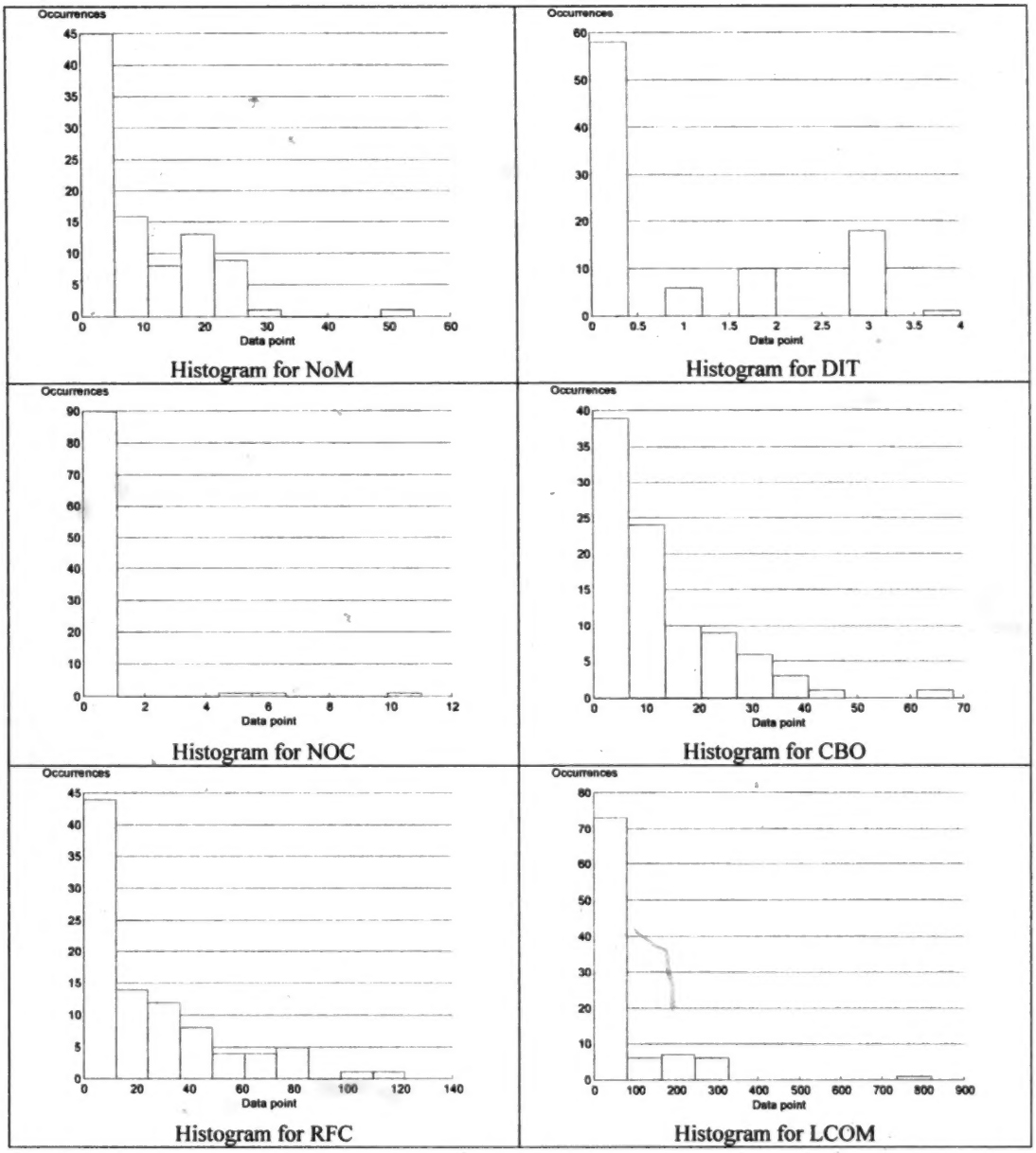


Table A 1: Histogram of the CK metrics for data set A

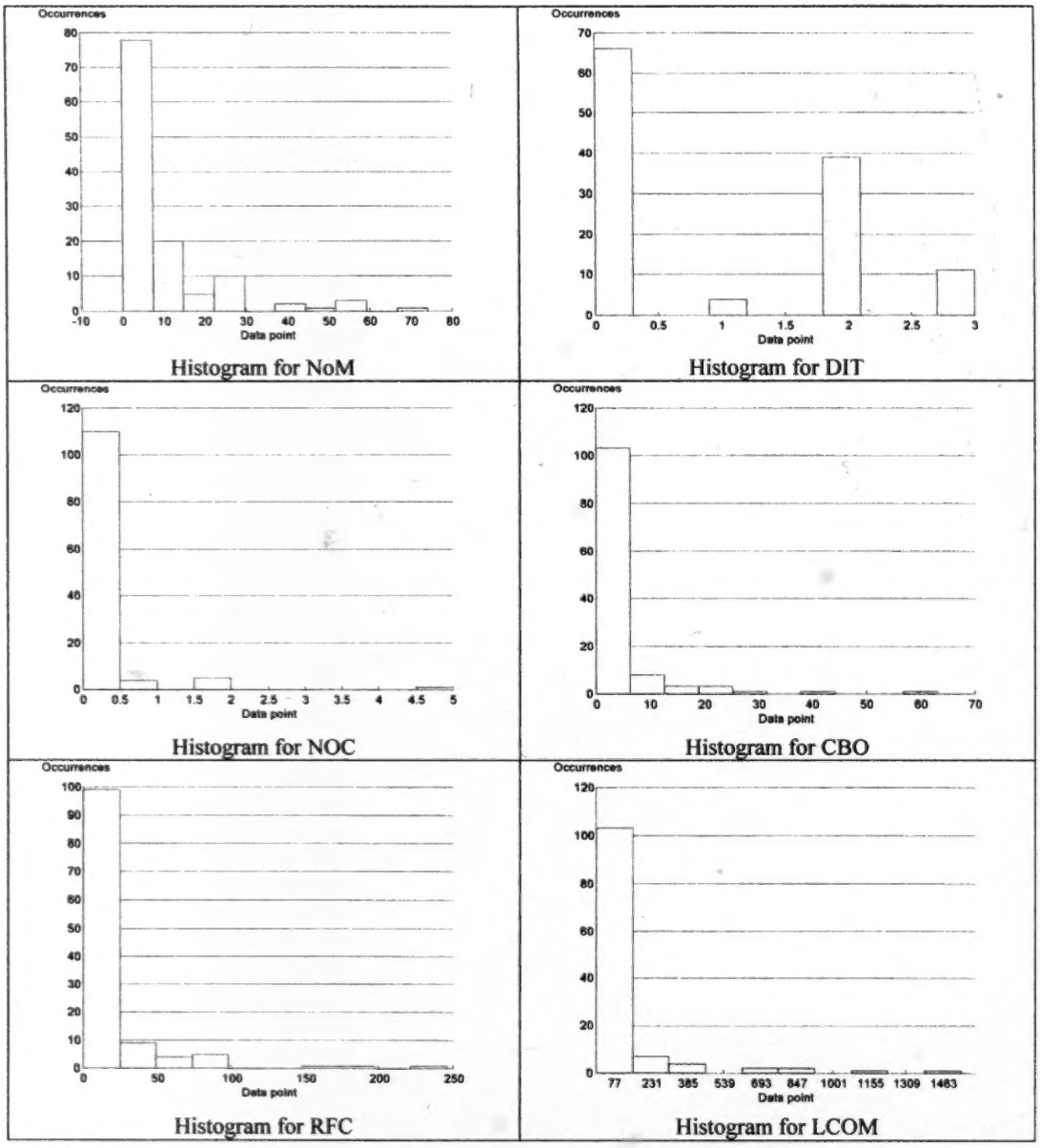


Table A 2: Histogram of the CK metrics for data set B

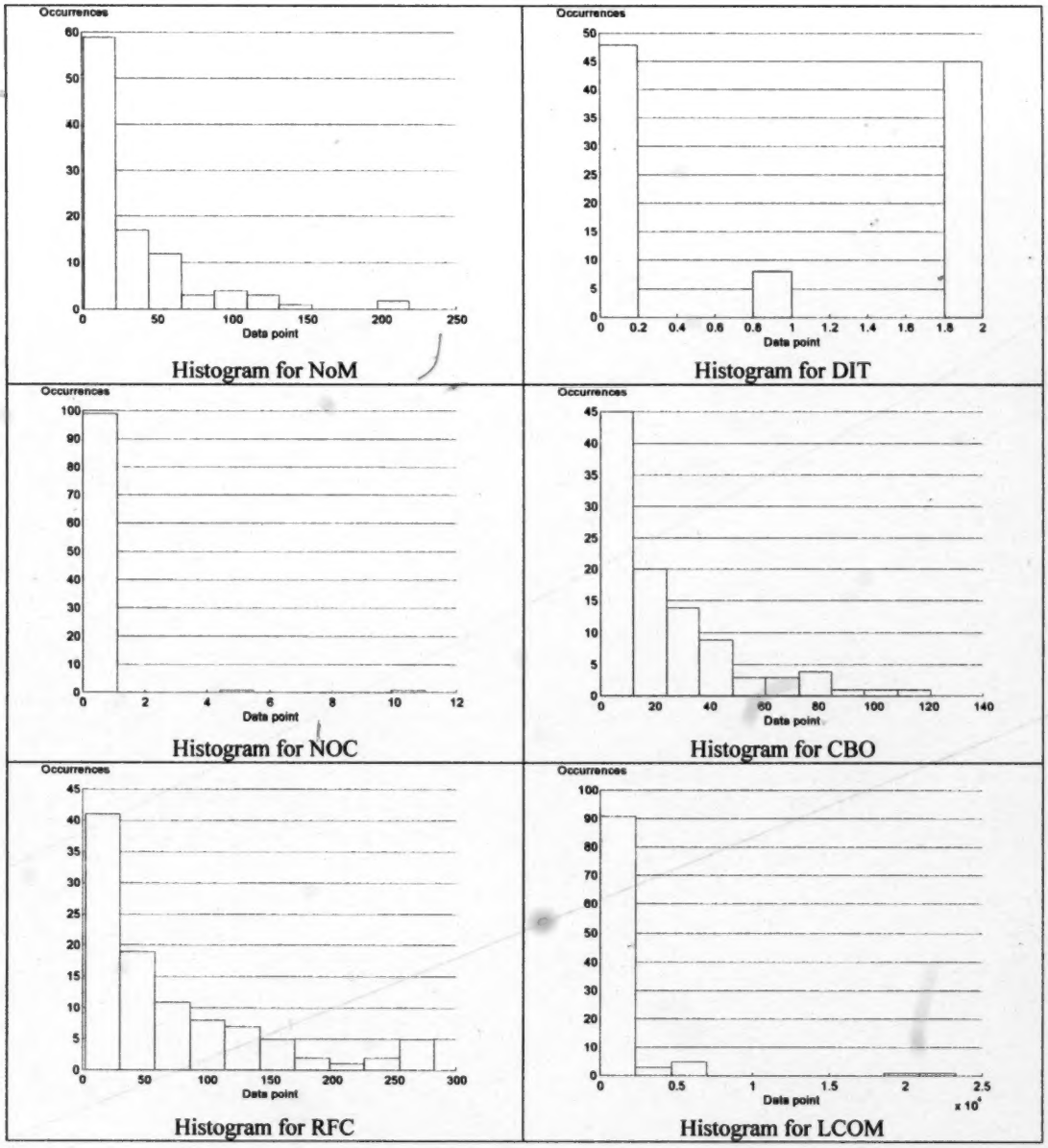


Table A 3: Histogram of the CK metrics for data set C

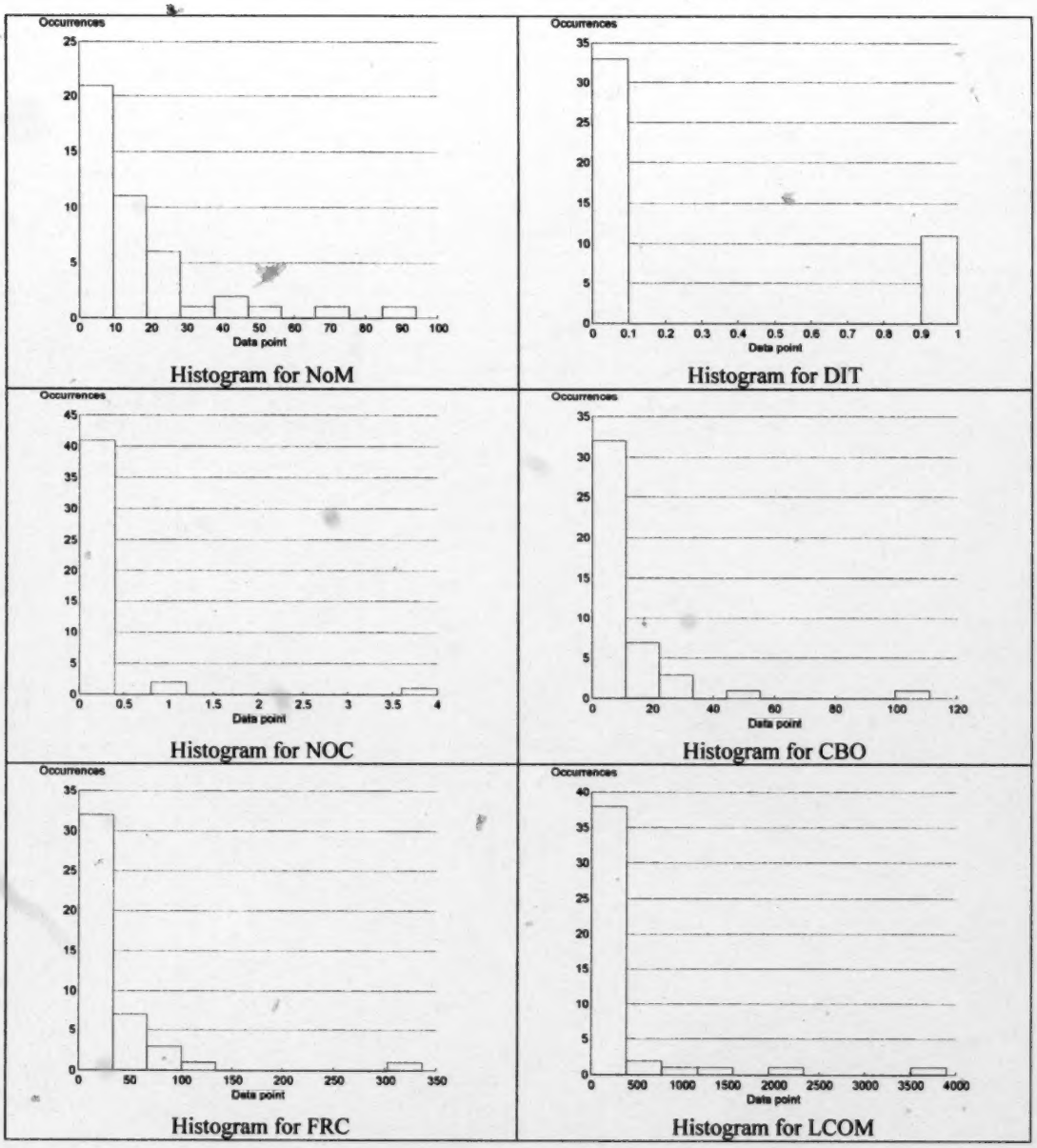


Table A 4: Histogram of the CK metrics for data set D

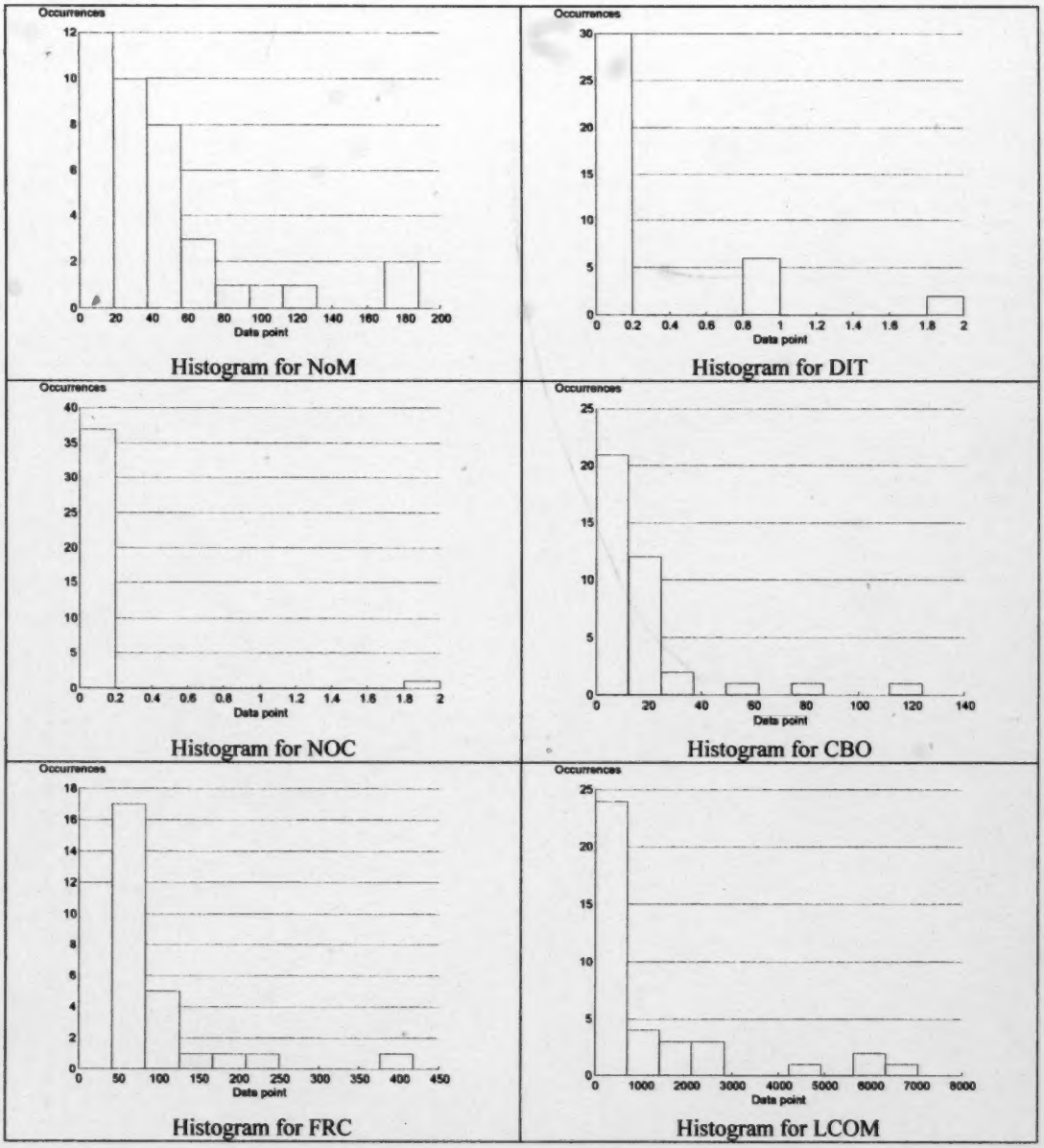


Table A 5: Histogram of the CK metrics for data set E

Appendix B

Results of Fuzzy c means clustering

Appendix B contains the plots of the results obtained in the clustering method. Table B 1 trough Table B 4 show the results of the clustering method to split the data into zero and non-zero elements, plots for 2, 4, and 6 clusters are presented. Project A is not shown since that data set does not contain zeroes in the output domain.

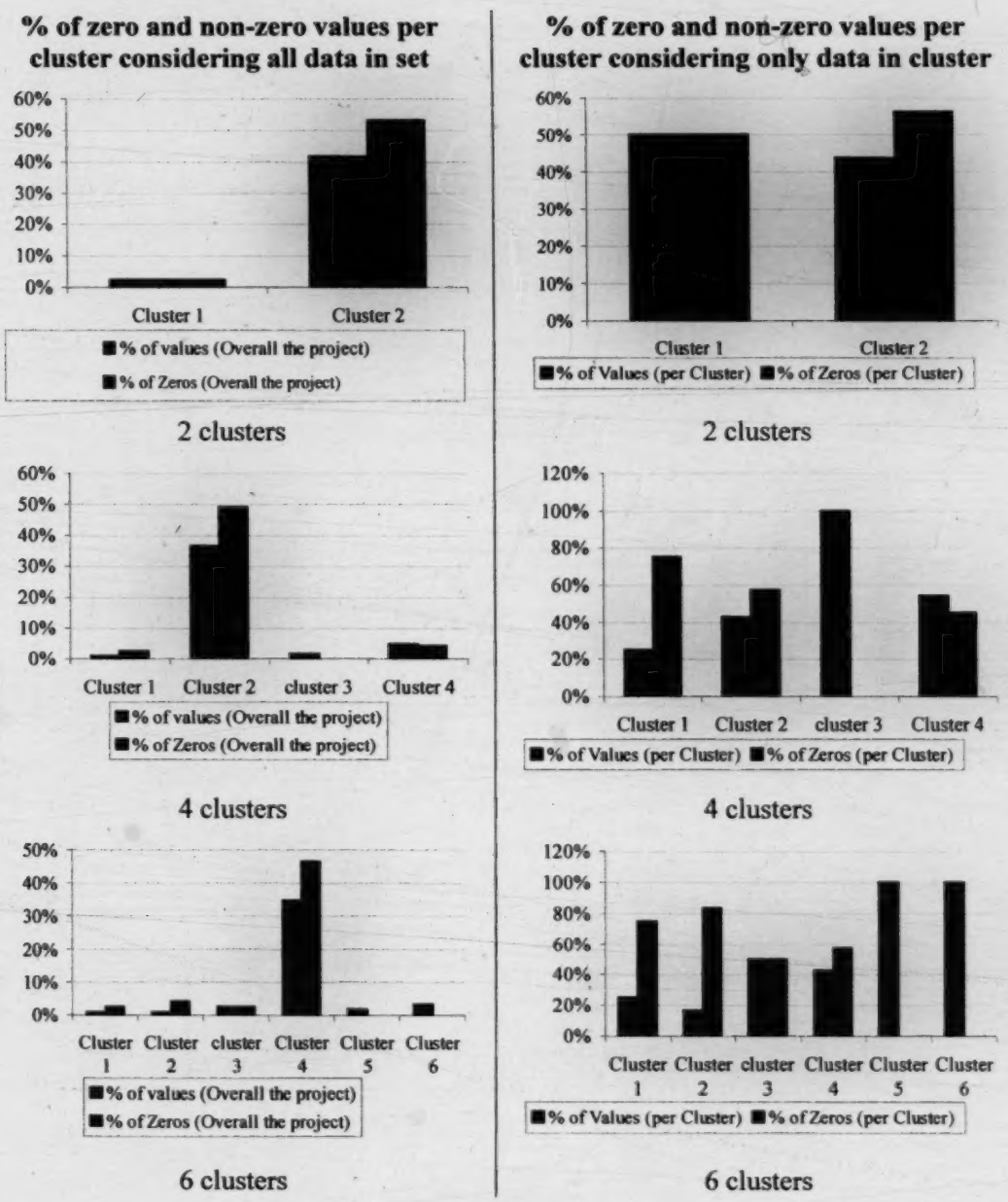


Table B 1: Zero data distribution for project B after clustering

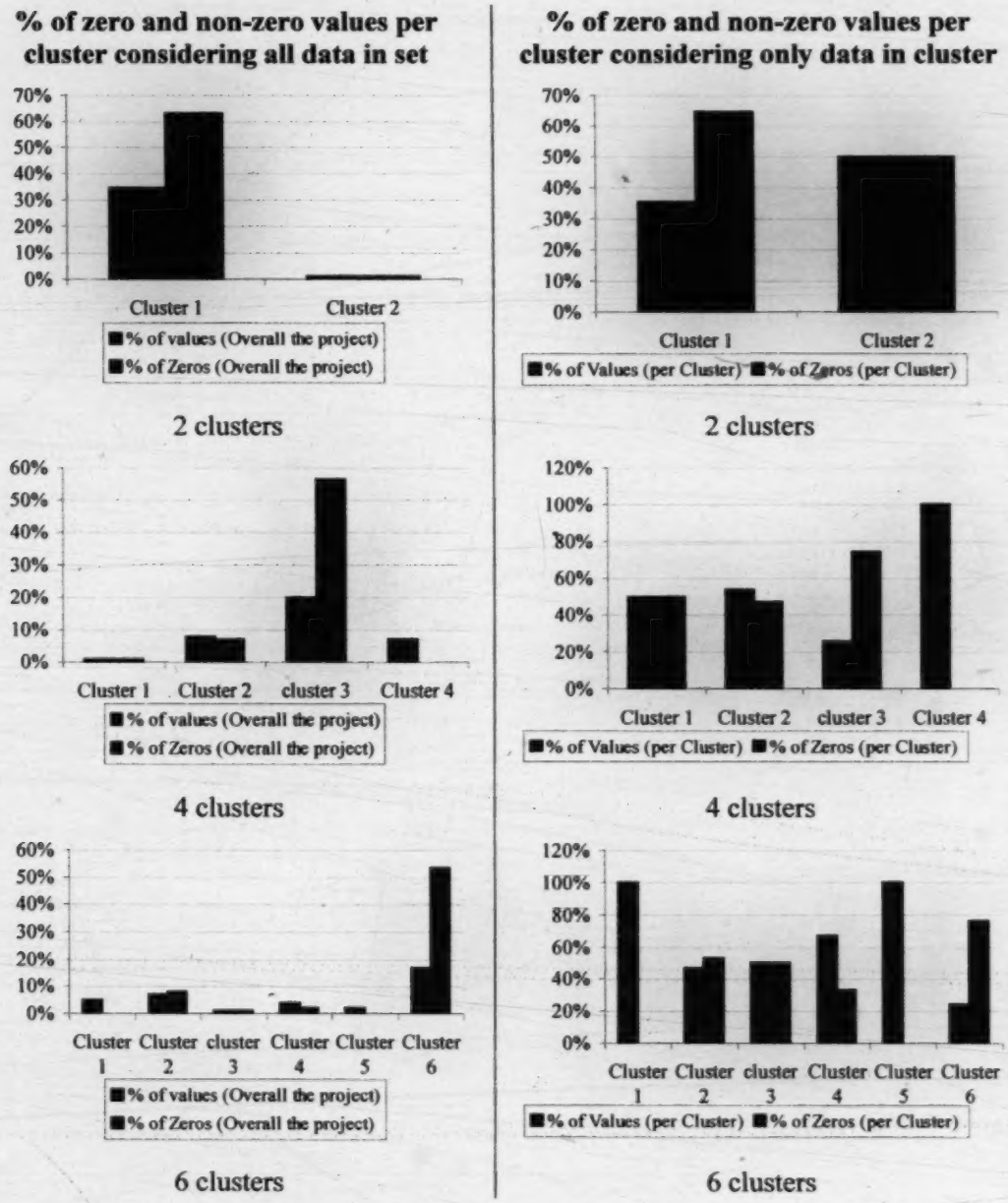


Table B 2: Zero data distribution for project C after clustering

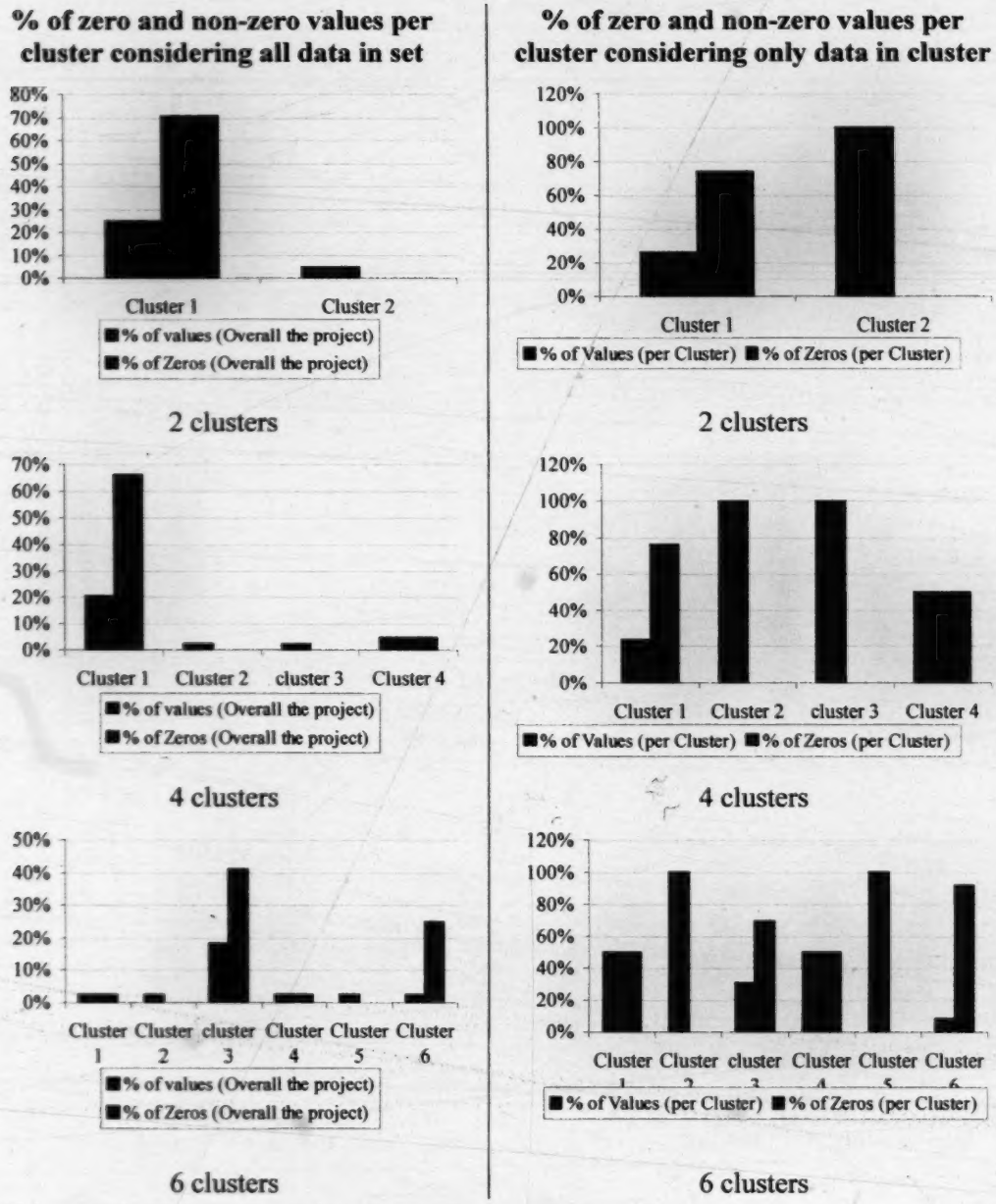


Table B 3: Zero data distribution for project D after clustering

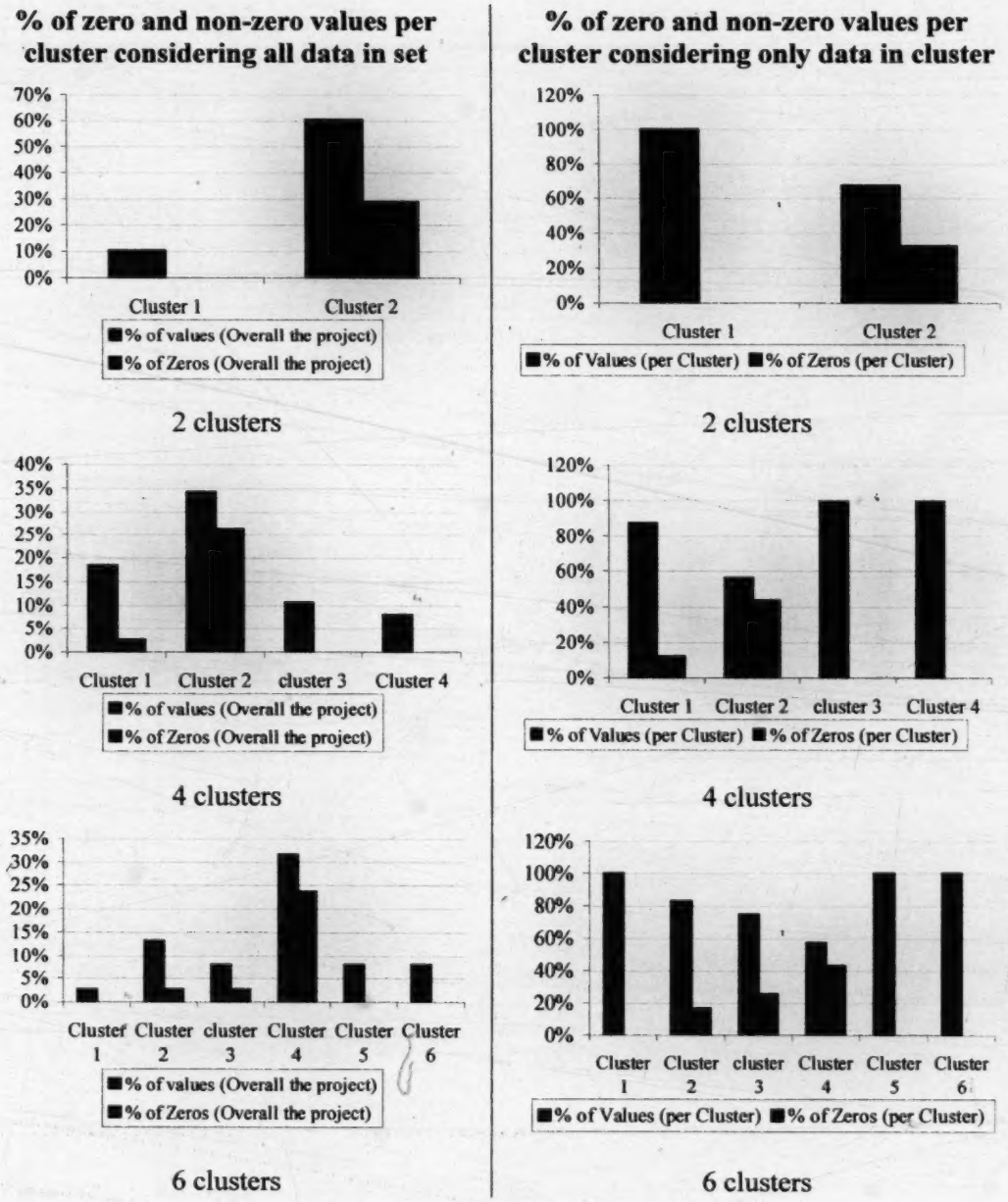


Table B 4: Zero data distribution for project E after clustering

Appendix C

Results of Clustering and local regression

Appendix C shows plots and tables related to the clustering and local regression algorithm. Table C 1 to Table C 13 show the regression coefficients of the linear models for each dimension in the clusters under study, if the value is "*inf*" then it means infinite value. Table C 14 to Table C 26 present the associated sum of errors for each model and the data in the corresponding cluster; the tables show the errors per dimension as well.

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5		cluster 6	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.041	0.145	0.062	0.124	0.121	0.113	0.200	0.119	0.692	0.083	0.048	0.115
2	-0.247	0.323	-0.134	0.222	-0.029	0.160	-0.188	0.140	-0.205	0.168	-0.135	0.126
3	-0.266	0.158	-0.232	0.142	-0.238	0.145	-1.161	0.143	-0.267	0.167	-0.035	0.124
4	0.186	0.095	0.166	0.087	0.022	0.135	-0.232	0.148	0.380	0.129	-0.246	0.147
5	0.160	0.105	0.176	0.083	0.091	0.114	0.017	0.134	0.558	0.105	-0.056	0.132
6	-0.007	0.154	-0.036	0.143	0.169	0.118	0.501	0.132	0.826	0.132	0.214	0.116

Table C 1: Regression coefficients for 6 clusters in project A

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.000	0.000	0.673	-0.127	1.900	-0.021	0.062	0.012	0.386	-0.017
2	0.000	0.000	0.920	-0.557	0.089	0.082	0.524	0.007	0.315	-0.003
3	Inf	Inf	Inf	Inf	-0.028	0.147	-0.049	0.016	-0.027	0.025
4	0.000	0.000	1.510	-0.035	1.560	0.044	0.257	0.008	0.581	-0.008
5	0.000	0.000	1.870	-0.182	2.920	0.006	0.220	0.009	0.560	-0.012
6	0.000	0.000	-0.853	0.289	7.670	0.067	-0.015	0.015	0.392	-0.003

Table C 2: Regression coefficients for 5 clusters in project B

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5		cluster 6	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	-0.299	0.151	-0.186	0.100	0.081	0.004	1.310	-0.004	-0.032	0.064	0.406	-0.004
2	0.500	0.015	0.129	0.026	Inf	Inf	0.190	0.005	0.112	0.010	0.154	-0.016
3	-0.199	0.080	-0.128	0.051	-0.021	0.007	-0.192	0.099	-0.075	0.063	-0.082	0.061
4	-0.047	0.076	0.049	0.040	0.316	0.000	1.720	0.008	0.707	0.025	0.570	0.008
5	-0.233	0.102	-0.048	0.053	0.235	0.002	2.680	-0.007	0.224	0.050	0.568	0.006
6	-0.787	0.135	-0.505	0.093	0.079	0.006	4.240	0.053	-0.183	0.064	0.360	0.026

Table C 3: Regression coefficients for 6 clusters in project B

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5	
Dim.	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.1200	0.0420	0.2300	0.0400	0.2100	0.0022	0.2500	0.0270	0.2500	0.0077
2	-0.1100	0.1400	inf	inf	0.1400	-0.0680	Inf	Inf	-0.0003	0.0220
3	-0.0560	0.1400	Inf	Inf	Inf	Inf	-0.0960	0.0440	-0.0500	0.0230
4	-0.2500	0.2600	0.4700	-0.0690	0.1800	-0.0022	0.2300	0.0190	0.1600	0.0038
5	0.2400	-0.0640	0.3400	-0.0610	0.1100	0.0092	0.2500	0.0160	0.1800	0.0039
6	0.0940	0.0740	0.7200	0.0540	0.7000	0.0130	1.3000	0.0340	1.6000	0.0140

Table C 4: Regression coefficients for 5 clusters in project C

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5		cluster 6	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.286	0.026	0.302	0.025	0.167	0.024	0.244	0.008	0.190	0.010	0.302	0.015
2	Inf	Inf	-0.089	0.045	-0.097	0.048	0.041	0.000	0.032	0.005	0.077	0.000
3	-0.096	0.044	-0.096	0.044	-0.096	0.044	Inf	Inf	0.053	0.031	Inf	Inf
4	0.516	-0.001	0.533	-0.002	0.185	0.017	0.127	0.012	0.090	0.015	0.408	-0.033
5	0.469	0.002	0.505	-0.001	0.182	0.015	0.129	0.007	0.092	0.012	0.275	-0.015
6	1.690	0.033	1.830	0.033	0.152	0.035	1.070	0.017	0.547	0.021	0.877	0.030

Table C 5: Regression coefficients for 6 clusters in project C

	cluster 1		cluster 2		cluster 3		cluster 4	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.0000	0.0000	0.1900	0.0400	0.3700	0.0310	0.1200	0.0870
2	Inf	Inf	Inf	Inf	Inf	Inf	0.1200	0.0490
3	Inf	Inf	Inf	Inf	-0.0970	0.0440	Inf	Inf
4	0.0000	0.0000	0.1600	0.0350	0.1100	0.0330	0.4100	-0.0370
5	0.0000	0.0000	0.1800	0.0270	0.5300	0.0066	0.2600	-0.0130
6	0.0000	0.0000	0.7400	0.0490	41.0000	0.0150	0.1100	0.1100
	cluster 5		**cluster 6**		**cluster 7**			
Dimension	*m*	*x*	*m*	*x*	*m*	*x*		
1	0.2600	0.0078	0.2200	0.0099	0.2700	0.0032		
2	inf	inf	-0.1000	0.1300	0.0630	-0.0320		
3	Inf	Inf	0.0500	0.0340	Inf	Inf		
4	0.0210	0.0310	0.0520	0.0260	0.1600	0.0040		
5	0.0870	0.0180	0.1000	0.0130	0.1400	0.0035		
6	1.4000	0.0140	0.5900	0.0230	1.5000	0.0094		

Table C 6: Regression coefficients for 7 clusters in project C

	cluster 1		cluster 2		cluster 3		cluster 4	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.0800	0.0400	0.4000	0.0500	0.0800	0.0700	0.1000	0.0200
2	Inf	Inf	0.2000	0.0500	Inf	Inf	Inf	Inf
3	-0.0900	0.0400	-0.2000	0.1000	-0.2000	0.0900	-0.0700	0.0300
4	0.3000	0.0200	0.6000	-0.0100	0.0800	0.0700	0.1000	0.0200
5	0.4000	0.0100	0.4000	0.0020	0.1000	0.0600	0.1000	0.0200
6	0.5000	0.0400	1.0000	0.0700	0.1000	0.0800	-0.0800	0.0300
	cluster 5		**cluster 6**		**cluster 7**		**cluster 8**	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	-0.0020	0.0300	0.1000	0.0300	0.2000	0.0100	0.2000	0.0100
2	-0.0800	0.0400	0.0300	0.0400	inf	inf	-0.1000	0.1000
3	-0.0700	0.0300	-0.1000	0.0400	Inf	Inf	0.0500	0.0300
4	0.1000	0.0200	0.1000	0.0200	0.0900	0.0200	0.0400	0.0300
5	0.0800	0.0200	0.0700	0.0300	0.1000	0.0100	0.0700	0.0200
6	-0.4000	0.0300	0.4000	0.0300	0.9000	0.0200	0.5000	0.0200

Table C 7: Regression coefficients for 8 clusters in project C

	cluster 1		cluster 2		cluster 3	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*
1	0.2140	0.0254	0.2840	0.0227	-0.1460	0.0759
2	inf	inf	-0.1640	0.1640	Inf	Inf
3	Inf	Inf	Inf	Inf	-0.0906	0.0679
4	0.2150	0.0365	-0.8960	0.2290	0.7030	0.0124
5	0.2130	0.0355	-0.2550	0.1560	0.9210	0.0059
6	0.2050	0.0422	0.3080	0.0737	-0.9400	0.0714

Table C 8: Regression coefficients for 3 clusters in project D

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.9640	0.0741	-0.1820	0.0527	-0.1660	0.0805	0.1300	0.0488	-0.1730	0.0625
2	Inf	Inf	Inf	Inf	Inf	Inf	0.1800	0.0699	-0.0109	0.0547
3	Inf	Inf	-0.0625	0.0469	-0.0801	0.0681	-0.1020	0.0870	Inf	Inf
4	-0.8000	0.4460	-0.5610	0.0633	0.0325	0.0607	0.1740	0.0593	-0.0991	0.0518
5	-1.4200	0.5570	-0.5110	0.0612	-0.7870	0.1070	0.1870	0.0553	-0.4210	0.0653
6	6.4400	0.0233	-3.6100	0.0533	-1.0300	0.0753	0.1800	0.0557	-2.5500	0.0657

Table C 9: Regression coefficients for 5 clusters in project D

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5		cluster 6	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	-0.775	0.118	0.091	0.074	0.856	0.055	-0.732	0.184	0.084	0.062	-0.651	0.130
2	Inf	Inf	0.161	0.089	Inf	Inf	Inf	Inf	Inf	Inf	-0.102	0.146
3	-0.250	0.063	Inf	Inf	Inf	Inf	Inf	Inf	-0.089	0.076	Inf	Inf
4	-1.390	0.130	0.136	0.080	0.054	0.163	-0.356	0.111	0.054	0.066	0.036	0.066
5	-2.300	0.175	0.148	0.076	0.333	0.133	-2.870	0.282	0.107	0.064	-1.080	0.113
6	-6.220	0.085	0.154	0.073	5.900	0.061	-2.170	0.121	-2.800	0.076	-4.100	0.103

Table C 10: Regression coefficients for 6 clusters in project D

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	0.000	0.156	1.310	-0.159	0.009	0.114	0.230	0.122	0.441	0.075
2	0.000	0.156	-1.060	0.594	0.610	0.101	0.217	0.137	0.536	0.138
3	0.000	0.156	-0.211	0.211	Inf	Inf	Inf	Inf	Inf	Inf
4	0.000	0.156	0.990	-0.023	0.499	0.059	0.886	0.026	0.502	0.089
5	0.000	0.156	2.170	-0.207	0.260	0.093	0.672	0.056	1.230	-0.012
6	0.000	0.156	0.922	-0.010	0.809	0.097	0.316	0.121	0.473	0.109

Table C 11: Regression coefficients for 5 clusters in project E

	cluster 1		cluster 2		cluster 3		cluster 4		cluster 5		cluster 6	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	-2.610	2.630	-0.955	0.172	0.502	0.075	0.754	0.044	0.577	0.066	0.898	0.047
2	-1.750	1.880	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	-0.159	0.211
3	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	Inf	-0.199	0.199
4	1.000	-0.005	0.493	0.028	0.521	0.102	0.636	0.082	0.552	0.100	0.742	0.068
5	1.730	-0.728	-0.425	0.127	1.760	-0.083	1.720	-0.052	1.690	-0.059	1.790	-0.047
6	-2.160	2.390	-2.370	0.128	0.429	0.125	0.865	0.093	0.493	0.120	0.795	0.121

Table C 12: Regression coefficients for 6 clusters in project E

	cluster 1		cluster 2		cluster 3		cluster 4	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*	*m*	*x*
1	-7.0000	1.9800	0.5200	0.0728	0.3250	0.1800	0.2030	0.0896
2	Inf	Inf	-0.4880	0.3750	-0.4010	0.3650	0.2240	0.0521
3	Inf	Inf	-0.2340	0.2340	Inf	Inf	Inf	Inf
4	3.9800	1.2300	0.8320	0.0715	2.3700	0.0259	1.3100	0.0435
5	1.3200	0.7610	1.1000	0.0270	1.1100	0.0849	0.4580	0.0755
6	-3.6400	0.6910	0.5480	0.0945	0.1110	0.2360	0.1520	0.1040

	cluster 5		cluster 6		cluster 7	
Dimension	*m*	*x*	*m*	*x*	*m*	*x*
1	-0.9550	0.1720	-0.0448	0.1200	0.2230	0.1080
2	Inf	Inf	Inf	Inf	0.1910	0.1280
3	Inf	Inf	Inf	Inf	Inf	Inf
4	0.4930	0.0281	0.4300	0.0571	0.9370	0.8540
5	-0.4250	0.1270	0.0272	0.1110	0.7020	0.0227
6	-2.3700	0.1280	0.1720	0.1040	0.3290	0.0991

Table C 13: Regression coefficients for 7 clusters in project E

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.0431	0.0480	0.0478	0.0175	0.0204	0.0243
2	0.3220	0.3140	0.3060	0.0322	0.0596	0.0255
3	0.0444	0.0407	0.0435	0.0308	0.0601	0.0425
4	0.0560	0.0607	0.0686	0.0229	0.0411	0.0194
5	0.0726	0.0796	0.0843	0.0204	0.0284	0.0274
6	0.0347	0.0347	0.0302	0.0288	0.0288	0.0186

Table C 14: Error for regression models in clusters of project A

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0.5690	0.0752	0.0677	0.0106	0.0309
2	0.0370	0.3790	0.4330	0.0026	0.0411
3	0.0000	0.0293	0.0927	0.0120	0.0425
4	0.0625	0.0037	0.0690	0.0048	0.0135
5	0.1800	0.0057	0.0702	0.0052	0.0146
6	0.0795	0.0338	0.0862	0.0045	0.0258

Table C 15: Error for regression models in clusters of project B

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.1200	0.1180	0.0037	0.0458	0.0412	0.0355
2	0.0141	0.0566	0.0008	0.2790	0.2790	0.2890
3	0.0352	0.0272	0.0083	0.0645	0.0306	0.0469
4	0.0394	0.0322	0.0015	0.0448	0.0200	0.0172
5	0.0408	0.0334	0.0020	0.0456	0.0255	0.0169
6	0.0275	0.0197	0.0009	0.0557	0.0283	0.0317

Table C 16: Error for regression models in clusters of project B

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0.3930	0.0799	0.0573	0.0150	0.0077
2	0.0686	0.8360	0.7200	0.0128	0.3650
3	0.2350	0.0469	0.0083	0.0174	0.0085
4	0.2300	0.1110	0.1000	0.0216	0.0199
5	0.4610	0.1990	0.1760	0.0221	0.0180
6	0.3040	0.0342	0.0035	0.0117	0.0030

Table C 17: Error for regression models in clusters of project C

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.0145	0.0142	0.0437	0.0129	0.0167	0.0488
2	0.0132	0.0249	0.0406	0.7950	0.7840	0.7580
3	0.0180	0.0180	0.0164	0.0041	0.0227	0.0293
4	0.0130	0.0127	0.0389	0.0355	0.0369	0.0648
5	0.0139	0.0133	0.0538	0.0533	0.0579	0.1120
6	0.0121	0.0121	0.0361	0.0027	0.0023	0.0215

Table C 18: Error for regression models in clusters of project C

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
1	0.0003	0.0269	0.0125	0.1710	0.0106	0.0156	0.0078
2	0.0000	0.0172	0.0140	0.5200	0.9330	0.9130	0.7890
3	0.0000	0.0172	0.0249	0.0683	0.0037	0.0298	0.0028
4	0.0006	0.0565	0.0159	0.1400	0.0326	0.0352	0.0199
5	0.0003	0.0586	0.0128	0.2870	0.0455	0.0546	0.0278
6	0.0000	0.0140	0.0139	0.1110	0.0022	0.0025	0.0018

Table C 19: Error for regression models in clusters of project C

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4
1	0.0111	0.0539	0.0836	0.0099
2	0.0117	0.1730	0.0247	0.0085
3	0.0203	0.0667	0.0333	0.0171
4	0.0132	0.0430	0.0691	0.0243
5	0.0123	0.0702	0.1020	0.0251
6	0.0116	0.0459	0.0686	0.0084
Dimension	Cluster 5	Cluster 6	Cluster 7	Cluster 8
1	0.0111	0.0398	0.0146	0.0191
2	0.0814	0.1530	0.9260	0.9130
3	0.0171	0.0192	0.0047	0.0269
4	0.0251	0.0746	0.0360	0.0376
5	0.0269	0.1170	0.0592	0.0648
6	0.0084	0.0099	0.0032	0.0027

Table C 20: Error for regression models in clusters of project C

Dimension	Cluster 1	Cluster 2	Cluster 3
1	0.0619	0.1490	0.0240
2	0.8880	0.3700	0.0205
3	0.0131	0.0969	0.0655
4	0.0573	0.1040	0.0134
5	0.0573	0.0901	0.0134
6	0.0586	0.0953	0.0200

Table C 21: Error for regression models in clusters of project D

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0.1500	0.0153	0.0235	0.1000	0.0155
2	0.2380	0.0135	0.0178	0.0493	0.5200
3	0.2380	0.0760	0.0768	0.0805	0.0126
4	0.2080	0.0148	0.0166	0.0501	0.0115
5	0.1830	0.0135	0.0169	0.0485	0.0114
6	0.1880	0.0134	0.0172	0.0619	0.0126

Table C 22: Error for regression models in clusters of project D

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.0209	0.1370	0.0757	0.0308	0.0187	0.0200
2	0.0186	0.0670	0.1100	0.0246	0.0198	0.7190
3	0.0234	0.0285	0.1100	0.0246	0.1020	0.0174
4	0.0205	0.0691	0.0980	0.0226	0.0176	0.0151
5	0.0186	0.0670	0.0880	0.0227	0.0170	0.0151
6	0.0186	0.0847	0.0908	0.0238	0.0196	0.0174

Table C 23: Error for regression models in clusters of project D

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0.0669	0.0229	0.0307	0.0981	0.0309
2	0.0244	0.2280	0.0305	0.1110	0.0443
3	0.0244	0.2780	0.0376	0.0838	0.0514
4	0.0116	0.0014	0.0244	0.0239	0.0295
5	0.0022	0.0197	0.0270	0.0388	0.0223
6	0.0509	0.0119	0.0323	0.0916	0.0332

Table C 24: Error for regression models in clusters of project E

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
1	0.5010	0.0293	0.0280	0.0219	0.0258	0.0313
2	0.5940	0.0263	0.0568	0.0557	0.0571	0.1130
3	0.3440	0.0263	0.0568	0.0557	0.0571	0.1370
4	0.0001	0.0195	0.0287	0.0287	0.0293	0.0328
5	0.0909	0.0230	0.0176	0.0167	0.0169	0.0296
6	0.4380	0.0255	0.0333	0.0304	0.0334	0.0420

Table C 25: Error for regression models in clusters of project E

Dimension	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
1	0.0321	0.0437	0.0490	0.0301	0.0293	0.0343	0.1170
2	0.0571	0.1680	0.1550	0.0818	0.0263	0.0311	0.1300
3	0.0571	0.2260	0.1050	0.0326	0.0263	0.0311	0.0860
4	0.0236	0.0186	0.0553	0.0177	0.0195	0.0197	0.0189
5	0.0194	0.0337	0.0463	0.0205	0.0230	0.0228	0.0384
6	0.0339	0.0386	0.0694	0.0319	0.0255	0.0278	0.1020

Table C 26: Error for regression models in clusters of project E

The following are the plots and regression coefficients for the multivariate regression models found with the clustering and local regression algorithm. The first table in the figures (top left) describes the regression coefficients for the specified cluster and the R^2 for the regression model, the second plot (top right) shows the normalized fixes Vs predicted *Y*, the third plot (lower left) depicts the standardize residuals, and the fourth plot (lower right) presents the histogram and distribution of the results.

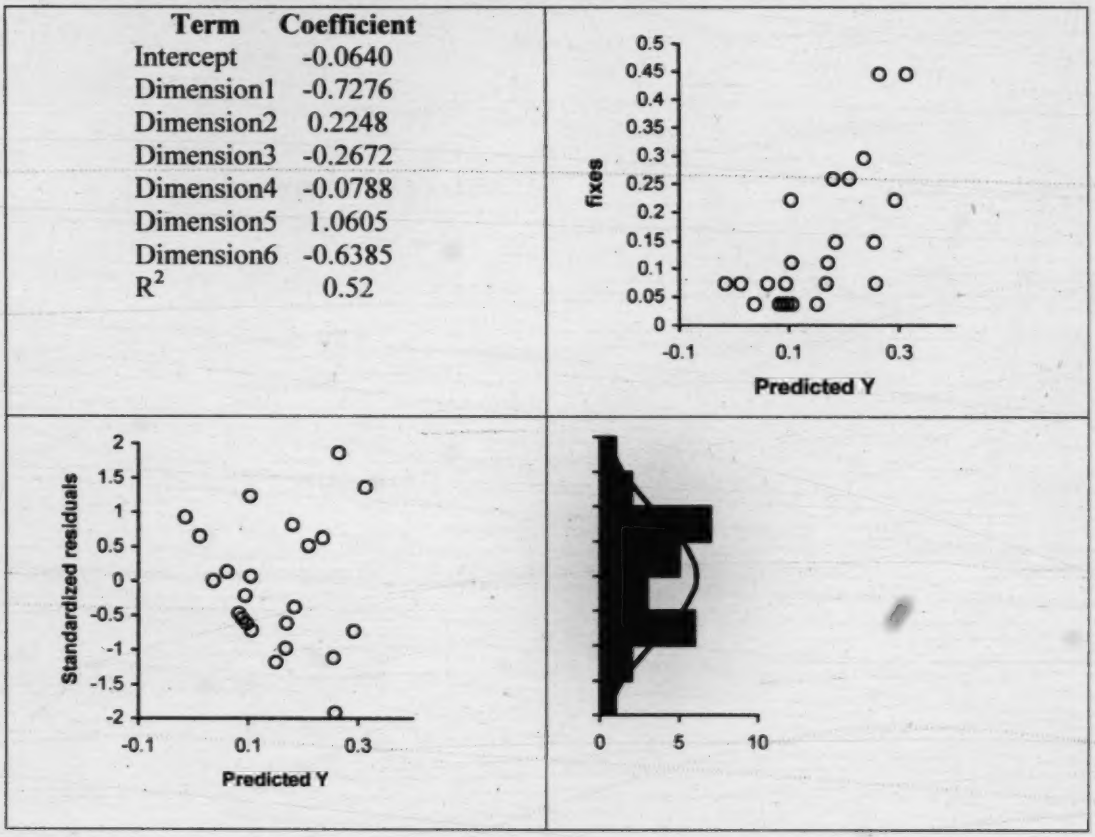
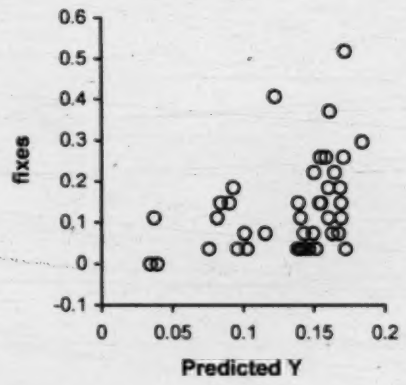
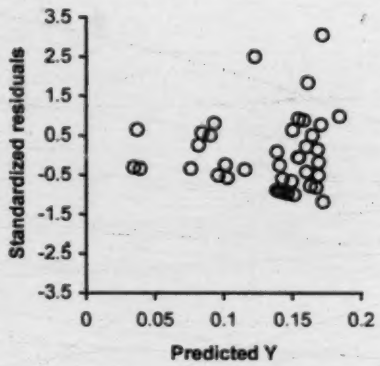


Figure C 1: Characteristics for the multivariate model in cluster 1 for project A

Term	Coefficient
Intercept	0.1631
Dimension1	0.1199
Dimension2	-0.2123
Dimension3	-1.4834
Dimension4	-0.5490
Dimension5	0.1400
Dimension6	-0.3240
R^2	0.12





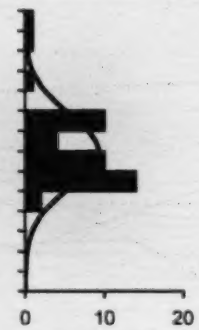
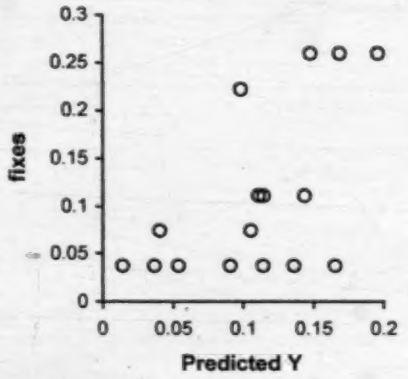


Figure C 2: Characteristics for the multivariate model in cluster 4 for project B

Term	Coefficient
Intercept	0.2200
Dimension1	-0.2969
Dimension2	-0.3672
Dimension3	-0.0136
Dimension4	-0.9103
Dimension5	0.4428
Dimension6	0.1604
R^2	0.34



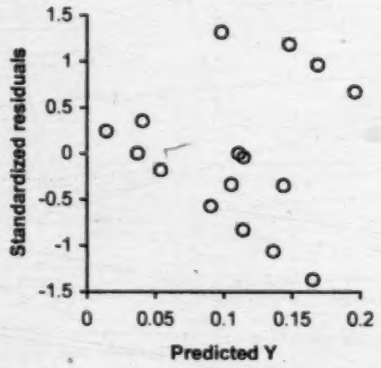
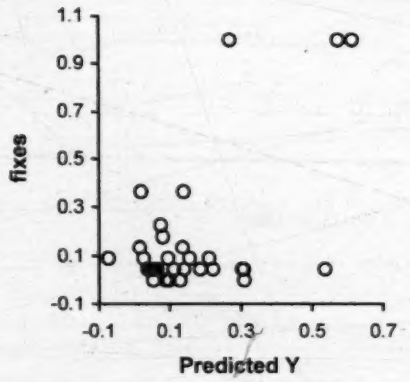
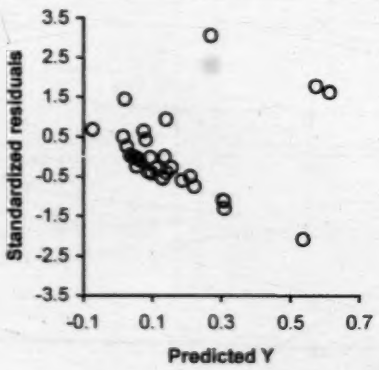


Figure C 3: Characteristics for the multivariate model in cluster 6 for project B

Term	Coefficient
Intercept	0.3095
Dimension1	-1.8265
Dimension2	-0.2969
Dimension3	0.1267
Dimension4	-2.3419
Dimension5	5.8360
Dimension6	8.6553
R^2	0.33





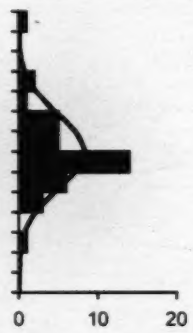
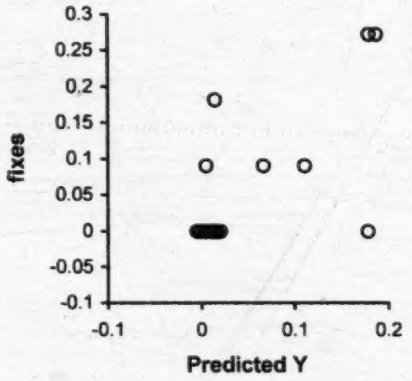


Figure C 4: Characteristics for the multivariate model in cluster 3 for project B

Term	Coefficient
Intercept	0.0008
Dimension1	0.0109
Dimension2	0.5223
Dimension3	-0.0022
Dimension4	0.3143
Dimension5	-0.0641
Dimension6	-0.0424
R^2	0.54



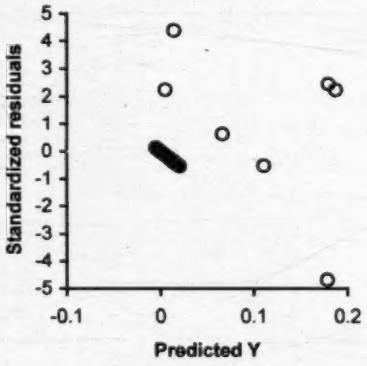


Figure C 5: 1 Characteristics for the multivariate model in cluster 4 for project B

Appendix D

Switching Regression Models and Fuzzy Clustering

Appendix D presents the results for the experiments described in chapter 7. The information provided in Tables F1 to F5 was obtained considering 2 clusters in the regression model algorithm; they show the *offset* and *slope* of the linear regression models and the overall error in each dimension.

Project A (2 Clusters)

Regression Line	Cluster	Slope	Offset	Error
1	1	0.273572	10.223602	0.349568
	2	0.060594	3.076264	
2	1	1.173620	10.502018	0.340702
	2	0.001662	3.189538	
3	1	-6.260869	8.260869	0.227734
	2	0.113048	2.337510	
4	1	0.349183	8.737515	0.371399
	2	0.043538	2.934638	
5	1	0.485146	7.005956	0.454323
	2	0.082704	2.012067	
6	1	0.02143	2.876745	0.127821
	2	1.042253	7.605633	

Table D 1: Regression models and error in project A

Project B (2 Clusters)

Regression Line	Cluster	Slope	Offset	Error
1	1	-0.368059	23.411907	0.358471
	2	0.103262	0.468515	
2	1	5.714452	0.242984	0.129220
	2	-6.038461	14.846153	
3	1	-0.181459	0.499013	0.302133
	2	-4.772277	11.287128	
4	1	0.0	22.0	0.277314
	2	0.161970	0.630472	
5	1	0.011084	1.979467	0.383446
	2	-0.262295	70.524590	
6	1	0.003815	1.233109	0.444479
	2	0.946725	1.576951	

Table D 2: Regression models and error in project B

Project C (2 Clusters)

Regression Line	Cluster	Slope	Offset	Error
1	1	1.132653	-0.040816	0.195821
	2	0.025407	0.385094	
2	1	1.25	6.5	0.169385
	2	0.375	0.102272	
3	1	-0.096551	0.482758	0.215829
	2	-0.537878	7.916666	
4	1	0.189508	4.372565	0.166934
	2	0.033448	0.044425	
5	1	0.072137	4.139857	0.172546
	2	0.011433	0.043737	
6	1	0.000218	1.338094	0.987342
	2	0.300620	-0.172025	

Table D 3: Regression models and error in project C

Project D (2 Clusters)

Regression Line	Cluster	Slope	Offset	Error
1	1	0.358573	3.799123	0.887290
	2	0.034616	-0.094198	
2	1	-6.666666	7.0	0.275767
	2	3.851851	0.148148	
3	1	-1.5625	6.25	0.749712
	2	-0.212121	0.212121	
4	1	6.0	-4.25	0.112073
	2	0.049505	0.058409	
5	1	0.012317	-0.019191	0.867509
	2	0.028859	6.110172	
6	1	0.000533	1.254481	0.691601
	2	0.030617	-0.101182	

Table D 4: Regression models and error in project D

Project E (2 Clusters)				
Regression Line	**Cluster**	**Slope**	**Offset**	**Error**
1	1	0.130301	11.788798	0.197722
	2	0.020447	1.946398	
2	1	-8.142857	15.214285	0.232103
	2	19.461538	3.038461	
3	1	0.463458	18.792546	0.187522
	2	1.264358	13.465287	
4	1	0.161454	10.724098	0.177577
	2	0.118789	1.094680	
5	1	0.022757	1.097732	0.189022
	2	0.051331	10.405285	
6	1	0.002528	13.428158	0.193524
	2	0.000504	2.273172	

Table D 5: Regression models and error in project E